

AD-A096 549

DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
A USERS MANUAL FOR A PDP-11 CROSS ASSEMBLER (CRASS) FOR THE TI--ETC(U)
MAR 81 I S ZARITSKY
DTNSRDC-81/010

UNCLASSIFIED

NL

| of |
AD
A096549



END
DATE
FILMED
4-81
DTIC



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER DTNSRDC-81/010	2. GOVT ACCESSION NO. ADA A096 549	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A USERS MANUAL FOR A PDP-11 CROSS ASSEMBLER (CRASS) FOR THE TI-9900 MICROPROCESSOR		5. TYPE OF REPORT & PERIOD COVERED Final
7. AUTHOR(s) Irving S. Zaritsky		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS David W. Taylor Naval Ship Research and Development Center Bethesda, Maryland 20084		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Supply Systems Command Research and Technology Division Washington, D.C. 20376		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (See reverse side)
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE March 1981
		13. NUMBER OF PAGES 90
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Microcomputer Assembly Language Microprocessor Assembler Cross Assembler		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes CRASS, a two-pass cross-assembler for the TI-9900 microprocessor. The assembler was written in GIRL-FORTRAN to be used on a PDP-11 computer with an RT-11 operating system. It uses the standard TI-9900 mnemonics and allows for most of the same assembly time directives. (Continued on reverse side)		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(Block 10)

Program Element 62760N
Project F60531
Task Area TF60531091
Work Unit 1800-008

(Block 20 continued)

CRASS has the following features:

- Symbolic memory addressing
- Portability - Since CRASS is written mostly in FORTRAN, it can be easily adapted to any machine having a FORTRAN IV compiler.
- Left to right arithmetic expression handling in binary, decimal, or hexadecimal
- Logical expression handling
- Specified program starting address; however, there is a default value.
- Less program assembly time than with the TI assembler since the code which is created is non-relocatable and therefore does not require a linkage step.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
<i>A</i>	

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

	Page
LIST OF TABLES	iv
ABSTRACT	1
ADMINISTRATIVE INFORMATION	1
INTRODUCTION	1
USING CRASS	2
GENERAL DISCUSSION	2
CONSTANTS AND EXPRESSIONS	3
Hollerith Constants	3
Numerical Constants	3
Identifiers	3
Logical and Numerical Operators	3
Logical and Numerical Expressions	4
INPUT FORMAT	4
OPERAND FORMATS	5
LIMITATIONS AND DIFFERENCES WITH THE TI ASSEMBLER	7
ERROR CODES	9
PROGRAM DESCRIPTION	10
DATA STRUCTURES USED BY CRASS	10
Instruction-Directive Tree	10
Identifier Tree	11
Label String	12
Token String	12
Item String	13
FORMING THE OBJECT CODE WORD	14
Instruction Code Skeletons	14
Object Code Formats	15
PROGRAM FLOW	16
Overview	16
Description and Example	16
BRIEF SUBROUTINE DESCRIPTIONS	19
Subroutine ADDREG	19
Subroutine LEXSCN	19
Subroutine ADDNAM (NODE, IDENT)	20
Subroutine INSTRU	20
Subroutine TOKSCN	20
Subroutine ITMSCN	21
Subroutine LINOUT	21

	Page
Subroutine LINEIN	21
Subroutine OPFLD (I1)	22
Subroutine TMPSTR (ISTLOC, LENGTH)	22
Subroutine VARLST	23
Subroutine COMPUT (OPRAND, OPRATR)	23
Subroutine DECHEX (DECNUM, HEX(1))	23
Subroutine FMT1 (ADRCOD(1))	24
Subroutine SRCLST (THISPC)	24
Subroutine ABSOUT	24
Subroutine ERROUT (THISPC)	24
Subroutine TWOCMP (HEX(1))	25
Function LVRTSH (WORD, BITS)	25
Function LVLFSH (WORD, BITS)	25
UPDATING CRASS	26
PROPOSED ADDITIONS AND IMPROVEMENTS	26
ACKNOWLEDGMENTS	26
APPENDIX A - GIRL-FORTRAN PROGRAM LISTING OF CRSGEN, THE DATA GENERATOR OF THE INSTRUCTION-DIRECTIVE TREE	27
APPENDIX B - GIRL-FORTRAN PROGRAM LISTING OF CRASS	31
APPENDIX C - VARIABLES IN LABELED COMMON	69
APPENDIX D - SAMPLE SET OF INSTRUCTIONS AND DIRECTIVES	73
APPENDIX E - SOURCE AND IDENTIFIER LISTING GENERATED FROM APPENDIX A AS INPUT	77
APPENDIX F - MACHINE CODE GENERATED FROM APPENDIX A AS INPUT	81
REFERENCES	83

LIST OF TABLES

1 - Instruction Operand Format Descriptions	5
2 - Instruction Operand Format Numbers	6
3 - Directive Operand Format Descriptions	7
4 - Line Status Error Codes	9

	Page
5 - Token Codes	12
6 - Item Type Codes	13
7 - Machine Code Formats	15

ABSTRACT

This report describes CRASS, a two-pass cross-assembler for the TI-9900 microprocessor. The assembler was written in GIRL-FORTRAN to be used on a PDP-11 computer with an RT-11 operating system. It uses the standard TI9900 mnemonics and allows for most of the same assembly time directives.

CRASS has the following features:

- Symbolic memory addressing,
- Portability - Since CRASS is written mostly in FORTRAN, it can be easily adapted to any machine having a FORTRAN IV compiler,
- Left to right arithmetic expression handling in binary, decimal, or hexadecimal,
- Logical expression handling,
- Specified program starting address; however, there is a default value,
- Less program assembly time than with the TI assembler since the code which is created is non-relocatable and therefore does not require a linkage step.

ADMINISTRATIVE INFORMATION

This work was performed in the Computer Science Division of the Computations, Mathematics, and Logistics Department under the sponsorship of NAVSUP 043C, Task Area TF60531091, Work Unit 1800-008.

INTRODUCTION

This report describes CRASS, a two-pass cross-assembler for the TI9900 microprocessor.¹ The assembler was written in GIRL²-FORTRAN to be used on a PDP-11 computer with an RT-11 operating system. It uses the standard TI9900 mnemonics and allows for most of the same assembly-time directives.

CRASS has the following features:

- Symbolic memory addressing
- Portability - Since CRASS is written mostly in FORTRAN, it can be easily adapted to any machine having a FORTRAN IV compiler.
- Left to right arithmetic expression handling in binary, decimal, or hexadecimal

*A complete listing of references is given on page 83.

- Logical expression handling
- Less program assembly time than with the TI assembler since the code which is created is non-relocatable and therefore does not require a linkage step.

Although program debugging would take less assembly time with relocatable code, the time difference is not considered significant for our purposes.

Several uses for the TI9900 microprocessor are envisioned. All these projects will require rapid software development. These projects include use of the TI microprocessor to:

- Control an "intelligent" Logistics Communication Terminal between the SNAP II* and NAVMACS** systems.
- Control the interface between a hardware associative memory and the PDP-11 computer.
- Provide the computing power for an experimental distributed microprocessor.
- Provide the interface for other similar man-machine and machine-machine projects (such as signature verification).

USING CRASS

GENERAL DISCUSSION

An assembly language program may be assembled on the PDP-11 by executing the CRASS.SAV file. If this file is on the system disk, the form is:

R CRASS

The program will respond by asking for the input and output file names:

"PLEASE ENTER FILE NAMES IN COMMAND STRING FORM"

The command string format for CRASS is:

"output file 1 [, output file 2] = input file"

The first output file contains the absolute object code and has a default extension name "ABS." The second output file is optional. It contains the source and identification listing if the "LIST" directive is included in the assembly program. This file has a default extension name "LST." Note that the RK1 disk unit must be turned on, since CRASS places one of its temporary scratch files there.

*Shipboard Nontactical ADP Program.

**Naval Modular Automated Communications System.

CONSTANTS AND EXPRESSIONS

Hollerith Constants

Hollerith constants are delineated by either a pair of quotes (') or a pair of dollar signs (\$), for example:

```
LABL1  BYTE  'A', 'X', $Y$;  
LABL2  DATA $AB$, 'XY';  
LABL3  TEXT  'ABCDE';  
LABL4  TEXT  $TODAY'S DATE IS$;  
LABL5  TEXT  'COST IS $5.00';
```

The following statements produce identical code:

```
BYTE 'A', 'B', 'C', 'D', 'E', 'F';  
DATA 'AB', 'CD', 'EF';  
TEST 'ABCDEF';
```

A discussion of the limitations on these directives may be found in the section on limitations and differences with the TI assembler and also in the TI9900 Manual,¹ Section 7, page 10.

Numerical Constants

Numerical constants may be expressed in decimal, binary, or hexadecimal form. Default is decimal; binary numbers are preceded by a percent sign (%) and hexadecimal numbers are preceded by a "greater than" operator (>). The minus sign (-) precedes either "%" or ">" for negative numbers. The following limitations are in effect:

```
Integer      + 32767  
Binary       +% 111 1111 1111 1111  
Hexadecimal  >7FFF, - >FFF
```

Identifiers

All identifiers must begin with an alphabetic character. The length of identifiers is limited (solely by the fixed-input format for labels) to six characters.

Logical and Numerical Operators

CRASS allows for the following logical operations:

<u>Operator</u>	<u>Function</u>
.+	Logical OR
.*	Logical AND
.-	Logical NOT (one's complement)
./	Modulo

Arithmetic functions are performed with the four standard operators:

+, *, -, and /.

Logical and Numerical Expressions

All expressions are evaluated strictly on a left to right basis with no parentheses allowed.

For example: X EQU 3+4*5;

X is set to 35, not 23.

Also, expressions must be resolved within two passes.

For example,

```
DATA X+3;
```

```
X    MOV    R1,R2;
```

will be correctly evaluated. However,

- 1) A EQU B+C
- 2) B EQU C+D
- 3) C EQU -10;
- 4) D EQU -5;

will not be correctly evaluated since "B" will not be resolved until the second pass examines statement number two.

Logical expressions follow the same structure and are evaluated in the same manner as numerical expressions.

For example:

```
X BYTE -F,+%1001.*10;
```

places "0A" into the appropriate byte and

```
Y BYTE -F.*%1001.+10;
```

places "0B" into the appropriate byte

INPUT FORMAT

Assembly statements consist of the following four fields:

1	8	13	
LABEL	INSTRUCTION OR DIRECTIVE	OPERAND FIELD	COMMENTS

The label may be placed anywhere in positions 1-6. Up to six alphanumeric characters are allowed, the first of which must be alphabetic. If an asterisk (*) is placed anywhere in the label field, the entire statement is treated as a comment.

Instructions and assembler directives must begin in position 8. Since the collection of instructions and directives available with CRASS is smaller than that offered by the TI assembler,¹ the reader is referred to Table 2 on page 6 and Table 3 on page 7 for instructions and directives available with CRASS. The reader is also referred to pages 6-18 through 6-60 and pages 7-8 through 7-14 from the TI Manual.¹

Operands may be placed anywhere from position 13 to position 72. Operand fields are separated by commas and the last field must be terminated by a semicolon (;). Comments may be placed after the semicolon. Instructions and directives which do not use the operand field do not require a semicolon. Arithmetic expressions must not contain any embedded blanks. Operand formats are discussed in the next section.

OPERAND FORMATS

The entire TI9900 instruction set requires twelve formats for the operand field. The format descriptions are listed in Table 1. The formats for the instruction set are summarized in Table 2. The formats for the directives are summarized in Table 3.

TABLE 1 - INSTRUCTION OPERAND FORMAT DESCRIPTIONS

Symbol	Description
S	Modifiable source address
D	Modifiable destination address
W	Unmodified workspace register
DIS	Displacement in bytes (displacement is in words for the TI assembler)
C	Count: integers 0-15
IOP	Immediate operand value
BIT	Integer 0-255

TABLE 1 (Continued)

Operand Format No.	Description
1	S,D
2	DIS
3	S,W
4	S,C
5	W,C
6	S
7	NONE
8	W,IOP
9	S,W (uses W_i and W_{i+1})
A	BIT
B	IOP
C	W

TABLE 2 - INSTRUCTION OPERAND FORMAT NUMBERS

Instr.	Format No.	Instr.	Format No.	Instr.	Format No.
A	1	JGT	2	RTWP	7
AB	1	JH	2	S	1
ABS	6	JHE	2	SB	1
AI	8	JL	2	SBO	A
ANDI	8	JLE	2	SBZ	A
B	6	JLT	2	SETO	6
BL	6	JMP	2	SLA	5
BLWP	6	JNC	2	SOC	1
C	1	JNE	2	SOCB	1
CB	1	JNO	2	SRA	5
CI	8	JOC	2	SRC	5
CLR	6	JOP	2	SRL	5
COC	3	LDCR	4	STCR	4
CZC	3	LI	8	STST	C
DEC	6	LIMI	B	STWP	C
DECT	6	LWPI	B	SWPB	6
DIV	9	MOV	1	SZC	1
IDLE	7	MOVB	1	SZCB	1
IN	6	MPY	9	TB	A
INC	6	NEG	6	X	6
INCT	6	NOP	7	XOP	9
INV	6	ORI	8	XOR	3
JEQ	2	OUT	6		

TABLE 3 - DIRECTIVE OPERAND FORMATS

Directive	Directive Number	Format
AORG	1	expression; (address)
BSS	3	expression;
BYTE	4	expression ₁ , . . . , expression _N ; N<19, if PC is odd N<20, if PC is even
DATA	5	expression ₁ , . . . , expression _N ; N<10
DXOP	8	format 6
END	9	none
EQU	10	expression;
EVEN	11	none
LIST	13	none
TEXT	17	Literal-list of N characters N<19, if PC is odd N<20, if PC is even
/	20	expression;
NOTES: 1. Slash is equivalent to the AORG directive. 2. Slash must be in column 8.		

LIMITATIONS AND DIFFERENCES WITH THE TI ASSEMBLER

The following (known) differences are listed in no particular order.

- 1) The input format for CRASS is quite rigid

Label - anywhere in columns 1-6

Instructions and Directives - must begin in column 8

Operand Field - anywhere in columns 13-72.

- 2) The jump instructions accept the value in the operand field as a quantity of bytes, whereas the TI assembler uses the operand value as a word quantity.

The operand value must be in the range -256 to +254 bytes.

- 3) Registers 0 through 15 must be referred to as "R0," "R1," . . . , "R15"

- 4) Hollerith (literal) data must be bounded by either a pair of dollar signs (\$) or a pair of apostrophies (').

- 5) The current location may be referred to, in CRASS, with an exclamation point (!). The TI assembler uses a dollar sign (\$).

6) DXOP. The format for this directive in CRASS is:

name DXOP extended operations number;

The format used by the TI assembler is:

DXOP name, extended operations number;

Note that the extended operation name should not be used as an ordinary label.

7) Assembly time constants and other operands must be resolved within two passes of the source program. For example, the following statements will be correctly resolved:

```
A EQU B;  
B EQU 2;
```

For the following statements, "A" would not be properly resolved:

```
A EQU B;  
B EQU C;  
C EQU 3;
```

8) To redefine the program counter (PC), both CRASS and the TI assembler will accept:

AORG absolute value;

CRASS will also accept a slash (in column 8):

/ absolute value;

9) Expression evaluation. Both assemblers evaluate expressions from left to right. However, on the TI assembler, a unary minus is performed first. For example,

Label1+Value1+(-Value2)

is legal with the TI assembler but not with CRASS.

10) Logical operators are legal in CRASS but do not appear to be in the TI assembler.

The operators are:

```
.+ Logical OR  
.* Logical AND  
.- Logical NOT (1's complement)  
./ Modulo
```

11) Individual BYTE, DATA, and TEXT statements are limited to a total of 20 characters in the operand field, 19 if the current value of the program counter (PC) is odd. The following statements are examples of the BYTE, DATA, and TEXT directives in which the operands contain the maximum number of allowable characters.

```

BYTE 1, 2, 3, 4, 5, 6, 7, 8, 9,
    0,>A,>B,>C,>D,>E,>F,
    'W','X','Y','Z';

```

```

DATA 'AB','CD','EF','GH',1J,'KL', 'MN','OP','QR','ST';

```

```

TEXT $ABCDEFGHJKLMNOPQRST$;

```

12) CRASS requires $133,200_8$ (46,720) bytes, or

$55,500_8$ ($23,360_{10}$) words plus space for the I/O buffers in order to execute on a PDP-11.

This memory requirement can be reduced should space become a consideration.

ERROR CODES

CRASS separates error conditions into three categories:

- 1) Multiple label definition
- 2) Nonexistent instruction name or assembler directive
- 3) All errors which originate in the operand field, such as:
 - a) Undefined memory references
 - b) Semantic errors, such as not using a workspace register as the source field operand for input format 5
 - c) Exceeding jump displacement limitations
 - d) Exceeding 20 characters for the DATA, BYTE, or TEXT directives.

When an error is caught on pass two, a message which describes the line status is sent to the terminal. The line status codes are given in Table 4.

TABLE 4 - LINE STATUS ERROR CODES

Error	Line Status
Label	2
Instruction	4
Operand Field	8

The line status is cumulative for each category of error found in a statement. Thus, a statement with a multiply-defined label and an incorrect instruction name will have a line status of $2 + 4 = 6$.

PROGRAM DESCRIPTION

DATA STRUCTURES USED BY CRASS

CRASS employs the Graph Information Retrieval System³ to create, store, and retrieve its key data structures. Both the Identifier and Op Code-Directives are stored in EPAM tree data structures.

These graphs are composed of source node - link - sink node triples as translated by GIRL.² Another way of referring to these triples is to say that the source node is related to the sink node by the link. For example, if the following relationships were to be inserted into a graph, using GIRL, they would appear as follows:

- 1) A is related to C by B
G A B C
- 2) A is related to C and D and E, by B
G A B (C, D, E)
- 3) A is related to C by B and C is related to E by D
G A B C D E
- 4) A is related to C, D, and E, by B and also D is related
to A by F
G A B (C, D F A, E)

Also discussed in this section are the "label," "token," and "item" (intermediate code) strings which are created in the first pass.

Examples:

- 1) Instruction-Directive Tree

This structure is created by a GIRL program called CRSGEN. It is executed before CRASS is run and need be re-executed only if either the instruction set or assembler directive set is changed. The entire program is listed in Appendix A. As an example, the GIRL statement representing the entire subgraph for directives and op codes beginning with the letter "D" is shown:

DATA	DIV
DEC	DORG
DECT	DXOP
DEF	

(Note that the dollar signs (\$) indicate that, although the particular node represents a unique state within the graph, it does not require a unique name.)

```

C
G      START D $ (A $ T $ A $ DIRECT "5,"
G      1      E $ (C $ (OPFIN("6,"'//06'),
G      2      T $ OPFIN("6,"'//06','//40')),
G      3      F $ DIRECT "6"))
G      4      (I $ V $ OPFIN("9','//3C'),
G      5      O $ R $ G DIRECT "7",
G      6      X $ O $ P $ DIRECT "8")

```

The root node for this graph is called "START." There are two types of terminal nodes. The terminal node for each directive (final link = "DIRECT") contains a unique identifying integer from 1 to 20. For example:

DATA	-	5
DEF	-	6
DORG	-	7
DXOP	-	8

The terminal node for each op code (final link = "OPFIN") describes:

- a) which of the twelve formats the operand field and object code may take, and
- b) a skeleton code which forms the basis for the object code. For example, the DIV instruction uses format nine and has a skeleton code of "3C." The statement:

DIV R2, R1;

produces "3C42" as the object code.

The nodes which are represented by dollar signs represent uniquely defined random nodes.

2) Identifier Tree

This graph is created largely during the assembler's first pass. When an identifier is first encountered, it is added to the graph, letter by letter. That is, each letter is a potentially new link in the graph.

If the identifier is first encountered in the label field, it is given a value equal to the current location of the program counter (PC) and its status is set to "defined-fullword." If the first encounter is in the operand field, the

identifier is given a value of zero and its status is set to "undefined." The root node for this graph is called "SYMBOL." The following GIRL statements represent a sample subgraph containing a highly correlated set of identifiers:

```

      BLACK          BLOCK4
      BLOCK          BLOCK5
      BLOCK1

G  SYMBOL  B  $ L $ (A $ C $ K $ STOP ("address," "status"),
G  1          0 $ C $ K $ (STOP ("address," "status"),
G  2          ONE  STOP ("address," "status"),
G  3          FOUR STOP ("address," "status"),
G  4          FIVE STOP ("address," "status"))

```

New "label," "token," and "item" strings are created for each statement during the assembler's first pass. After each statement is examined during the first pass, the label and token strings are destroyed, but the item string is saved on a file as the intermediate code for that statement.

3) Label string

The label string is composed of a "multivalued list" (MVL) as described by Zaritsky,³ pages 2-3. The source (root) node is called "STRING" and the link "LABELS." For example, the label "BLOCK1" has the following label string:

```
G  STRING LABELS (B,L,O,C,K,1)
```

4) Token string

The token string is the assembler's first attempt at codifying the operand field. The operand field is broken up into "tokens:" arithmetic operators [+ - * /], numbers, alphanumerics, literals, separators [, ; ()] and special characters [! @ % > .]. The token string consists of three separate MVL's:

a) The first MVL consists of a source node/link pair, "TOKEN, STRING," and a set of sink nodes consisting of a sequence of token code numbers representing the operand field. The token codes are listed in Table 5.

TABLE 5 - TOKEN CODES

8	+	16	NUMBER
9	*	17	ALPHANUMERIC
10	-	18	@
11	/	19	%
12	.	20	(
13	!	21	.
14	;	22	
15	LITERAL		

For example, the operand field:

@TABL->1A(R1),*R2+;

will result in the following token code string:

G TOKEN STRING (18, 17, 10, 20, 16, 21, 17, 12, 9, 17, 8, 14)
 [@, TABL, -, >, 1A, (, R1, , *, R2, +, ;]

b) The second MVL describes each token's position in the input buffer. It consists of a source node/link pair, "TOKEN, BUFPOS." For the preceding example, if the at-sign were placed in column 1, the MVL would appear as follows:

G TOKEN BUFPOS ("1","2","6","7","8","10","11","14","15","16","18","19")

c) The third MVL describes the length of each token. Although this string could have been eliminated and computed from the second MVL, the time savings warranted its inclusion. For the preceding example, the MVL would appear as follows:

G TOKEN LENID ("1","4","1","1","2","1","2","1","1","2","1","1")

5) Item String

The token string is reduced to a string of "items" which include:

- arithmetic operators [+ - * /]
- logical operators [.+ .- .* ./]
- literals
- separators [, ;]
- numbers
- identifiers
- register types [Ri *Ri @ Ri *Ri+]
- "current PC" operator [!]
- at-sign [@]

The item codes are listed in Table 6.

TABLE 6 - ITEM TYPE CODES

0	Reg	10	-
1	*Reg	11	/
2	@Reg	12	,
3	*Reg+	13	!
4	.+ logical OR	14	;
5	.* logical AND	15	literal
6	.- logical NOT	16	number
7	./ modulus	17	identifier
8	+	18	@
9	*		

The item string consists of four separate MVL's:

a) The first MVL consists of a source node, link pair "ITEM, ITMTYP" and the sink nodes consist of a sequence of item codes derived from the token string. The item codes are taken from Table 6.

The token string from the preceding example would result in the following MVL:

```
G   ITEM ITMTYP   ("18", "17", "10", "16", "0", "12", "3", "14")
                        [@   TABL   -   >1A   R1   ,   *R2+   ;]
```

b) The second MVL is the "ITEM,VALUE" pair.

The sink nodes contain the following types of information:

registers	- register number
identifiers	- assigned address or 0 if undefined
numbers	- decimal value
literals	- first two characters in ASCII format
operators	- 0
separators	- 0
exclamation point	- 0

In the example being used, the MVL (during pass two) would appear as follows:

```
G   ITEM VALUE ("0", "address of TABL", "0", "26", "1", "0", "2", "0")
```

c) The third MVL, "ITEM,BUFPOS", is identical to the "TOKEN,BUFPOS" string.

d) The fourth MVL, "ITEM,LENID", is identical to the "TOKEN,LENID" string.

FORMING THE OBJECT CODE WORD

Instruction Code Skeletons

The TI9900 microprocessor instructions take up one 16-bit word (four contiguous hexadecimal integers). Associated with each instruction is an instruction code skeleton, described in the 9900 Family Systems Design Book,¹ pages 6-18 through 6-60. This skeleton may consist of one to four hexadecimal integers which form the heart of each resultant object code word. CRASS takes the skeleton and modifies it as per the operand field to create the object code word. For example: The instruction DIV has a skeleton instruction code "3C₁₆" (0011 1100₂) and an operand format of 9. It will be shown shortly that the operand field "R2,R1;" translates to "042₁₆." Therefore, the statement: DIV R2,R1; translates to "3C42" for the final object code.

Note that an object (data) word is created for immediate operands and also that each memory reference requires an object (data) word.

Object Code Formats

Although there are twelve operand field formats, for the purposes of this assembler there are only eight different object code formats, which are listed in Table 7.

TABLE 7 - MACHINE CODE FORMATS

Operand Format	Operand Format Number	Machine Code Format
1 a) S,D	1	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC RC _d R _d RC _s R _s
b) S,W	3,9	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC R _d RC _s R _s
c) S,C	4	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC C RC _s R _s
2 a) DIS	2	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0001 JC DIS
b) BIT	A	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0001 IC BIT
3 W,C	5	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC C W
4 S	6	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC RC _s R _s
5 none	7	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC
6 W,IOP	8	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC W
7 IOP	B	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC
8 W	C	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 IC W
Explanation of abbreviations: IC - Instruction Code JC - Jump Code W - Workspace register C - Count S - Source D - Destination R _s , R _d - Register reference, if set to zero, the next word contains a memory reference. DIS - Displacement RC - Register Code R1 = 00 ₂ *R1 = 01 ₂ @R1 = 10 ₂ *R1+ = 11 ₂		

PROGRAM FLOW

Overview

CRASS requires two passes to create non-relocatable, absolute code. A program listing is given in Appendix B and the variables used are described in Appendix C. The major functions of each pass are:

- First pass - Assign address values to labels
Reduce operand field to an item string
(intermediate code)
Obtain instruction format and skeleton
Attempt to resolve EQU statements
- Second pass - Second attempt to resolve EQU statements
Create and output absolute code
Provide source and identifier listing if desired

Description and Example

To facilitate the program description which follows, the statement:

```
XYZ  MOV  @ABC->23C(R2),*R13+;    (comments)
```

will be converted to object code as an example.

Before CRASS examines any assembly code:

- 1) User and scratch files are defined
- 2) The data graph which describes the instructions and directives is read in.
- 3) The identifiers R0, R1, . . . , R15 are defined as registers.

Begin pass one:

- 1) Data strings and line statement error flags are re-initialized.
- 2) The ASCII input line is lexically scanned by subroutine LEXSCN.
 - a) The label is converted to a string.
G STRING LABELS (X,Y,Z)
 - b) The instruction is converted to a string.
G STRING OPCOD (M,0,V)
 - c) The operand field is converted to a string
(Note that the capitalized names are from labeled common /GIRLCH/).

```

G   TOKEN (STRING (ATSIGN, VARABL, MINUS, GTTHAN, NUMBER, LPAR, VARABL,
G   1          RPAR, COMMA, STAR, VARABL, PLUS, SCOLON),
G   2   BUFPOS ("1", "2", "5", "6", "7", "10", "11",
G   3          "13", "14", "15", "16", "19", "20"),
G   4   LENID  ("1", "3", "1", "1", "3", "1", "2",
G   5          "1", "1", "1", "3", "1", "1"))

```

- 3) If the statement contains a label, Subroutine ADDNAM is called to test for multiple definition and to add the label to the identifier tree if this was the first occurrence of the label. The label is then placed on a special scratch file.

```

G   SYMBOL X $ Y $ Z $ STOP ("address," "status")

```

where address = the current value of the PC and
status = "defined"

- 4) Subroutine INSTRU then tries to match the "STRING, OPCOD" list with either a valid instruction or an assembler directive from the Op code - Directive graph. If it is a directive, a unique identifying integer (from 1 to 20) is returned. If it is an instruction, an operand field format and a skeleton code are returned. For the instruction "MOV" these are

format #1
skeleton "C0₁₆"

- 5) Subroutine TOKSCN then converts the token string to an item string. The item string is the intermediate code for transition from pass one to pass two. The item string for this example is:

@, variable, minus, number, register[type 0], comma,
register[type 3], semicolon

Note that the values for the ITEM, STRING come from Table 6.

```

G   ITEM (STRING ("18", "17", "10", "16", "0", "12", "13", "14"),
G   VALUE ( 0, address, 0, decimal value, reg. no., 0, reg. no., 0),
G   BUFPOS ( . . . ),
G   LENID ( . . . ))

```

TOKSCN also assigns source and destination register codes. For this example, TOKSCN "sees"

@R2,*R13+

resulting in

$$R_s = 2 = 0010_2$$

$$R_d = 13 = 1101_2$$

$$RC_s = 2 = 10_2$$

$$RC_d = 3 = 11_2$$

- 6) ITMSCN is then called to create the machine code words for this statement. From Tables 2, 1, and 7, and page 6-19 of the TI 9900 Family Systems,¹ the first machine code word would be:

IC	RC _d	R _d	RC _s	R _s			
0123	45	6789	10 11	12	13	14	15
1100	11	1101	1 0	0	0	1	0
C	F	6		2			

which equals: "CF62₁₆." If the identifier "ABC" has been defined (that is, it occurred in the label field of a statement prior to this one) as, for example, "23F₁₆," then the memory reference is computed as 23F₁₆ - 23C₁₆ = 3. Therefore, the object code created is:

CF62
0003

If the statement containing "ABC" as a label did not occur until after the present one, the memory reference would have to be resolved in the second pass.

- 7) The intermediate code is then placed on a scratch file by Subroutine LINOUT.

Before pass two is begun, the read/write pointers of the two scratch files containing the intermediate code and the label names are reset to the beginning.

Begin pass two:

- 1) Data Strings and line statement error flags are re-initialized.
- 2) Subroutine LINEIN is called to read in the intermediate code for the next statement.
- 3) Subroutine ITMSCN is called for a second attempt to complete the machine code. If there are no errors, all memory references and "EQU," "DATA," and "BYTE" directives will be resolved at this time.

- 4) An error checking routine, ERRROUT, is then called to determine whether any line statement error flags have been turned on and, if so, to report any errors.
- 5) If requested by the user (assembler directive "LIST"), Subroutine SRCLST will output a source code listing of the current statement.
- 6) The object code is then placed on an output file by Subroutine ABSOUT.

After pass two is completed, if "LIST" is requested, the scratch file containing the label names is examined and Subroutine VARLST creates an identifier listing. Otherwise, the program is finished.

BRIEF SUBROUTINE DESCRIPTIONS

The main (driving) program was described in the previous section on program flow. It calls the following subroutines:

ITMSCN	LINEIN
ADDREG	ITMSCN
LEXSCN	ERRROUT
ADDNAM	SRCLST
INSTRU	ABSOUT
TOKSCN	VARLST
ASSIGN[RT-11 system library routine]	LINOUT

With a minimum of detail, this section describes the eighteen subroutines and two functions in CRASS.

Subroutine ADDREG

Function:

To add the identifier names R_0, R_1, \dots, R_{15} to the identifier graph and declare them to be workspace registers.

Called By: Main Routine

Subroutine Called: ADDNAM

Subroutine LEXSCN

Function:

To perform a lexical scan of each input statement to create:

- 1) Label string
- 2) Instruction string
- 3) Token string to represent the operand field.

The tokens would consist of arithmetic operators, numbers, alphanumerics, literals, separators, and special characters.

Called By: Main Routine

Subroutine ADDNAM (NODE, IDENT)

Function:

To search the identifier graph for the requested variable from the NODE, IDENT string and add it to the graph if not found. ADDNAM checks for multiple definition of a label and, if the label is new, saves it on a scratch file so that an identifier listing can be created at the end of the program.

Called By: Main Routine OPFLD
 ADDREG TOKSCN

Subroutine INSTRU

Function:

To search the Opcode - Directive graph for the requested instruction. If it is not found, an error flag is set and a return is made. If an assembler directive is matched, a unique integer identifying that directive is returned. This integer is used later by Subroutine ITMSCN. If an instruction is matched, its machine code skeleton and its operand field format number are returned.

Called By: Main Routine

Subroutine TOKSCN

Function:

To reduce the token string to an item string as described in the sections on data structures and program flow and also to assign values to these items. The items consist of:

<u>Item</u>	<u>Item Value</u>
Identifiers	- Address value or 0 if undefined
Registers	- Register number and code
Numbers	- Decimal value
Literals	- First two ASCII characters
Arithmetic operators	- 0
Logical operators	- 0
Separators	- 0
Exclamation mark	- 0
At-sign	- 0

Called By: Main Routine

Subroutines Called: TMPSTR TWOCMP
ADDNAM LVLFSH

Subroutine ITMSCN

Function:

To scan the item string, left to right, to create the final machine code.

- a) First Pass - The number of bytes (halfwords) needed for each instruction is determined and the PC is updated by that amount to assign address values to labels. To update the PC, the following assembler directives are examined:

AORG	EVEN
BSS	TEXT
BYTE	"/"
DATA	

The following directives are also examined on the first pass:

DXOP
END
EQU

- b) Second Pass - A second attempt is made to resolve EQU directives. Also, the machine code is constructed and placed on an output file.

Called By: Main Routine

Subroutines Called: FMT1 LVRTSH
OPFLD LVLFSH
DECHEX

Subroutine LINOUT

Function:

To place the intermediate code (item string) onto a scratch file during the first pass.

Called By: Main Routine

Subroutine LINEIN

Function:

To read the intermediate code into core for processing in pass two.

Called By: Main Routine

Subroutine OPFLD (I1)

Function:

To examine a single operand field as delineated by:

beginning to comma
beginning to semicolon
comma to comma
comma to semicolon

where I1 is the location in the operand buffer of the lefthand delineator.

For example:

```
      1      1  
1  5  0  5  
@TABL1(R3),*R2+;
```

To examine the source (left) operand, I1 is set to 0. To examine the destination (right) operand, I1 is set to 11. OPFLD attempts either to extract a register number or to compute a decimal value for the field. The decimal value is converted to hexadecimal and placed into a special array.

OPFLD also determines whether an extra object code word must be allotted for a memory reference.

Called By: ITMSCN

Subroutines Called: DECHEX TMPSTR
COMPUT ADDNAM

Subroutine TMPSTR (ISTLOC, LENGTH)

Function:

To take a sequence of "LENGTH" ASCII characters from the operand field beginning at location ISTLOC, and convert it to a GIRL multivalued list (MVL). For example,

Operand Field:

```
      1      1      2      2  
1  5  0  5  0  5  
@ABC->123(R1),@TABL1(R2);
```

If ISTLOC = 16,
LENGTH = 5,

the resultant MVL will be:

G STRING STRING (T,A,B,L,ONE)

Called By: TOKSCN
OPFLD

Subroutine VARLST

Function:

If the "LIST" option is requested, to place the status for each identifier and register on the source listing output file.

Called By: Main Routine

Subroutine Called: DECHEX

Subroutine COMPUT (OPRAND, OPRATR)

Function:

To take operands from the two-word array OPRAND () and perform a computation on:

OPRAND(1) OPRATR OPRAND(2)

as determined by the arithmetic or logical operator in OPRATR. The result is placed in OPRAND(1). If this routine is called more than once for a single operand field, the effect is one of left-to-right (no precedence) expression handling. For example, if Subroutine OPFLD is called to handle

,3+5*6

COMPUT will be called twice, first to compute 3+5 and then to compute 8*6.

Called By: OPFLD

Subroutine DECHEX (DECNUM, HEX(1))

Function:

To convert the decimal value in DECNUM to a four-digit hexadecimal number. The four digits are converted to ASCII characters and placed in the four-byte array, HEX().

Called By: ITMSCN ABSOUT
 OPFLD SCRLST
 VARLST ERROUT

Subroutine called: TWOCMP

Subroutine FMT1 (ADRCOD(1))

Function:

To take the source and destination register values and the source and destination register codes as input and (except for the instruction code) create the machine code for output format number one. This code is placed in the three-byte array, ADRCOD().

Called By: ITMSCN

Subroutine SRCLST (THISPC)

Function:

If the "LIST" directive is requested, to place the following information on an output file:

- Input Line number
- Address in both decimal and hexadecimal (THISPC)
- Line status (non-zero indicates error)
- Object Code
- Input statement
- Comments (if any)

A sample set of instructions and directives is given in Appendix D. The source and identifier listing created by this subroutine from the instructions and directives of Appendix D are given in Appendix E.

Called By: Main Routine

Subroutine Called: DECHEX

Subroutine ABSOUT

Function:

To place the object code on an output file. This routine is called during pass two for each statement which is not a comment. See Appendix F for the format used for generated object code.

Called By: Main Routine

Subroutine Called: DECHEX

Subroutine ERROUT (THISPC)

Function:

To report to the terminal during the second pass any line statement with an error. A single variable, LNSTAT, is used to describe any errors.

If error free,	LNSTAT = 0
If label error,	LNSTAT = 2
If instruction error,	LNSTAT = 4
If operand error,	LNSTAT = 8

For any combination of errors, the value of LNSTAT is cumulative. For example, if a single line has both an instruction and an operand field error, the line status would be reported as "12."

Called By: Main Routine

Subroutine called: DECHEX

Subroutine TWOCMP (HEX(1))

Function:

To convert a hexadecimal number (from HEX()) of four ASCII digits to its two's complement form and place it, again in ASCII form, back into the four-byte array HEX().

Called By: TOKSCN
DECHEX

Function LVRTSH (WORD, BITS)

Function:

To perform a right logical shift. The content of WORD is moved by the number of bits in BITS.

Called By: ITMSCN

Function LVLFSh (WORD, BITS)

Function:

To perform a left logical shift. The content of WORD is moved by the number of bits in BITS.

Called By: TOKSCN
ITMSCN

UPDATING CRASS

An understanding of GIRL² is required to update (or correct any errors in) CRASS.

CRASS consists of two source files:

- 1) CRASS1.GRL - all routines contain a mixture of GIRL and FORTRAN statements. The first line of code defines the GIRS buffer size.
- 2) CRASS2.FOR - all routines consist entirely of FORTRAN.

If CRASS1.GRL is modified, it must be converted to all-FORTRAN by the GIRL preprocessor.² It must then be compiled with the following switch:

```
.R FORTRAN
*CRASS1 = CRASS1/N:7
```

No special switches are needed when CRASS2.FOR is compiled:

```
*CRASS2 = CRASS2
```

A copy of the GIRS³ object file GIRS.OBJ is required to link the files into a "SAV" file:

```
.R LINK
*CRASS=CRASS1,CRASS2,GIRS,SYSLIB/F
```

Note that CRASS1.OBJ must be the first object file in the linking sequence.

If the set of instructions or assembler directives is to be modified, CRSGEN.GRL must be accessed. This program generates the graph which describes the entire legal set of instructions and assembler directives. It is written in GIRL and is listed in Appendix A. Of course, any change in the generator must be matched in the CRASS1 semantics routine ITMSCN. Also, since the GIRS buffer size is defined here, it must be identical to the first line of code from CRASS1.GRL.

PROPOSED ADDITIONS OR IMPROVEMENTS

- 1) More precise diagnostics
- 2) More flexible input format
- 3) Removal of the limitation for resolving assembly time constants (EQU directives) in two passes.
- 4) Allow parenthesis precedence in expression evaluation
- 5) Removal of, or at least improvement on, the 20 character limitation for the BYTE, DATA, and TEXT statements.

ACKNOWLEDGMENTS

The contributions and helpfulness of J. Carlberg, of DTNSRDC Code 1824, are gratefully acknowledged.

APPENDIX A
GIRL-FORTRAN PROGRAM LISTING OF CRSGEN, THE DATA
GENERATOR OF THE INSTRUCTION-DIRECTIVE TREE

```

900      NOSAVE, PRINT, COMMENTS
G      DEFINE EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
G      1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
G      2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
G      3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
G      4 ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
G      5 V, W, X, Y, Z, START, OPFIN, DIRECT, TOKODE, VARABL, NUMBER, LITRAL
C
G      EXECUTE
      CALL ASSIGN(99, 'SY: CRASS. GRF', 12)
C
C SET TOKEN CODES FOR ALL TOKENS WHICH MAY
C BE THE FIRST CHARACTER OF AN "ITEM"
G      TOKODE PLUS      "8"
G      TOKODE STAR      "9"
G      TOKODE MINUS     "10"
G      TOKODE SLASH     "11"
G      TOKODE COMMA     "12"
G      TOKODE EXCLAM    "13"
G      TOKODE SCOLON    "14"
G      TOKODE LITRAL    "15"
G      TOKODE NUMBER    "16"
G      TOKODE VARABL    "17"
G      TOKODE ATSIGN    "18"
G      TOKODE PERCNT    "19"
G      TOKODE GTTHAN    "20"
G      TOKODE LPAR      "21"
G      TOKODE PERIOD    "22"
C
C STORE DIRECTIVE VALUES AND INSTRUCTION FORMATS AND SKELETON CODES
C
G      START A $'ATEMP (OPFIN("1", '//A0'),
G      1          B $ (OPFIN("1", '//B0'),
G      2          S $ OPFIN("6", '//07', '//40'))
G      ATEMP      (I $ OPFIN("8", '//02', '//20'),
G      1          N $ D $ I $ OPFIN("8", '//02', '//40'),
G      2          O $ R $ G $ DIRECT "1")
C
G      START B $'BTEMP (OPFIN("6", '//04', '//40'),
G      1          E $ S $ DIRECT "2",
G      2          L $ (OPFIN("6", '//06', '//80'),
G      3          W $ P $ OPFIN("6", '//04'))
G      BTEMP      (S $ S $ DIRECT "3",
G      1          Y $ T $ E $ DIRECT "4")
C
G      START C $'CTEMP (OPFIN("1", '//80'),
G      1          B $ OPFIN("1", '//90'),
G      2          I $ OPFIN("8", '//02', '//80'))
G      CTEMP      (L $ R $ OPFIN("6", '//04', '//C0'),
G      1          O $ C $ OPFIN("3", '//20'),
G      2          Z $ C $ OPFIN("3", '//24'))
C
G      START D $'DTEMP (A $ T $ A $ DIRECT "5",
G      1          E $ (C $ (OPFIN("6", '//06'),
G      2          T $ OPFIN("6", '//06', '//40')),
G      3          F $ DIRECT "6"))
G      DTEMP      (I $ V $ OPFIN("9", '//3C'),
G      1          O $ R $ G $ DIRECT "7",
G      1          X $ O $ P $ DIRECT "8")

```



```

C
G   START S $ STEMP OPFIN("1", '//60')
G   STEMP B $ (OPFIN("1", '//70'),
G   1       D $ OPFIN("10", '//1D'),
G   2       Z $ OPFIN("10", '//1E'))
G   STEMP E $ T $ D $ OPFIN("6", '//07')
G   STEMP L $ A $ OPFIN("5", '//0A')
G   STEMP O $ C $ (OPFIN("1", '//E0'),
G   1       B $ OPFIN("1", '//F0'))
G   STEMP R $ (A $ OPFIN("5", '//08'),
G   1       C $ OPFIN("5", '//0B'),
G   2       L $ OPFIN("5", '//09'))
G   STEMP T $ (C $ R $ OPFIN("4", '//34'),
G   1       S $ T $ OPFIN("12", '//02', '//CO'),
G   2       W $ P $ OPFIN("12", '//02', '//A0'))
G   STEMP W $ P $ B $ OPFIN("6", '//06', '//CO')
G   STEMP Z $ C $ (OPFIN("1", '//40'),
G   1       B $ OPFIN("1", '//50'))
C
G   START T $ (B $ OPFIN("10", '//1F'),
G   1       E $ X $ T $ DIRECT "17",
G   2       I $ T $ L $ DIRECT "18")
C
G   START U $ N $ L $ DIRECT "19"
C
G   START X $ (OPFIN("6", '//04', '//80'),
G   1       D $ (P $ OPFIN("4", '//2C'),
G   2       R $ OPFIN("3", '//2B'))))
C
G   START SLASH $ DIRECT "20"
C
G   CALL LVDUMP(0,0,99)
C
G   WRITE(99) EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
G   1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
G   2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
G   3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
G   4 ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
G   5 V, W, X, Y, Z, START, OPFIN, DIRECT, TOKODE, VARABL, NUMBER, LITRAL
C
G   COMPLETE
/  COMPLETE

```

APPENDIX B
GIRL-FORTRAN PROGRAM LISTING OF CRASS

```

900      *99, NOSAVE COMMENTS
REAL*4 DEFEXT(2), SCRACH(3)

C
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1      DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

C
COMMON /ASC/      LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/   LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/   DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/   INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1      LSTFLG
COMMON /ASSEMB/   PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1      MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2      OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/    FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/   EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1      LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2      ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3      NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4      ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5      V, W, X, Y, Z
COMMON /GIRL/     STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1      ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2      VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3      OPFJN, DIRECT, START, TOKODE
COMMON /ASCII/    BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z

C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60), FILL(100)
DIMENSION ASCGRL(58), FILSPC(39)

C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
EQUIVALENCE (ASCGRL(1), EXCLAM), (HEXCOD(1, 1), BUF(1))

C
DEFINE STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
G 1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
G 2 LOGOR, LOGAND, LOGNOT, MODULO
C

DATA LINENO, PC, ERROR, FIRST, SECOND /0, 256, 0, 1, 2/
DATA BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
1 /1H , 1H0, 1H9, 1HA, 1HF, 1HZ/
DATA HEXTBL /1H0, 1H1, 1H2, 1H3, 1H4, 1H5, 1H6, 1H7, 1H8, 1H9,
1 1HA, 1HB, 1HC, 1HD, 1HE, 1HF/
DATA DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG, DIRFLG
1 /1, 2, 4, 8, 16, 32, 64, 128, 256/
DATA DEFEXT/6CRASSABS, 3RLST/
DATA SCRACH/3RRK1, 6RUSER , 3RTMP/

C
EXECUTE
G CALL ASSIGN(99, 'SY: CRASS GRF', 12)
CALL ASSIGN(14, 'SY: USER VAR', 11, 'SCR')

C
CALL ASSIGN(10, 'SY: USER ASS', 11)
C CALL ASSIGN(12, 'SY: USER ABS', 11)
C CALL ASSIGN(13, 'SY: USER LST', 11)

```

```

C      TYPE 2
2      FORMAT( ' PLEASE ENTER FILE NAMES IN COMMAND STRING FORM' )
      FILSPC(7)=0
      IF( ICSI( FILSPC, DEFEXT, , , 0) .NE. 0) STOP 'INVALID COMMAND STRING'
      IF( IASIGN(12, FILSPC(1), FILSPC(2), FILSPC(5), 0) .NE. 0) STOP 'INVALID
1      IASIGN 1'
      IF( FILSPC(7) .EQ. 0) GO TO 5
      IF( IASIGN(13, FILSPC(6), FILSPC(7), FILSPC(10), 0) .NE. 0) STOP 'INVALID
1      IASIGN 2'
5      IF( IASIGN(10, FILSPC(16), FILSPC(17), FILSPC(19), 32) .NE. 0) STOP
1      'INVALID IASIGN 3'
C
      IF( IASIGN(11, SCRACH(1), SCRACH(2), -1, 2) .NE. 0) STOP ' BAD SCRATCH
1      FILE REQUEST'
C
      READ(99) EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1      LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2      ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3      NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4      ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5      V, W, X, Y, Z, START, OPFIN, DIRECT, TOKODE, VARABL, NUMBER, LITRAL
C
      PASS = FIRST
      LABERR = .FALSE.
      ENDFLG = .FALSE.
      LSTFLG = .FALSE.
C
C      BEGIN NAME TABLE BY CREATING REGISTER NAMES: RO, ... , R15
      CALL ADDREG
C
C      READ IN NEXT LINE OF CODE
10     LINENO = LINENO + 1
C
      READ(10,1) LINE
      FORMAT(72A1)
C
C      SET LINE STATUS FLAGS
      TYPERR = .FALSE.
      LABERR = .FALSE.
      DIRERR = .FALSE.
      OPERR = .FALSE.
      INSTER = .FALSE.
      COMNTS = .FALSE.
      RESOLV = .TRUE.
      WRDNUM = 1
      BYTNUM = 1
      LNSTAT = 0
      DIRNUM = 0
C
C      CLEAR ASCII BUFFERS AND G1RL STRINGS
      DO 20 I1=1, 40
20     BUF(I1) = BLANK

```



```

C
Q      STRING-(LABELS,OPCOD)
Q      TOKEN -(STRING,BUFPOS,LENID)
Q      ITEM  -(ITHTYP,VALUE,BUFPOS,LENID)
C
C BREAK LINE STATEMENT UP INTO TOKEN STRINGS
      CALL LEXSCN
C
C IS THIS ENTIRE LINE A COMMENT?
22     IF(COMNTS) GO TO 40
C
C DOES THIS STATEMENT CONTAIN A LABEL?
Q      STRING+LABELS/30
C
C ADD LABEL TO NAME TREE AND ASSIGN CURRENT PROGRAM COUNTER (PC)
C AS ITS VALUE
      NODE = STRING
      IDENT = LABELS
      CALL ADDNAM(NODE, IDENT)
C
C EXAMINE INSTRUCTION
30     CALL INSTRU
C
C CONVERT OPERAND FIELD FROM A TOKEN STRING TO AN ITEM STRING
C (INTERMEDIATE CODE)
      CALL TOKSCN
C
C DETERMINE CORRECT NUMBER OF TARGET WORDS OF ABSOLUTE CODE NEEDED FOR
C THIS INSTRUCTION AND UPDATE PC BY THAT AMOUNT. THIS IS NEEDED IN THE
C FIRST PASS IN ORDER TO ASSIGN CORRECT ADDRESSES TO LABELS.
C DO NOT SCAN ITEM STRING IF INSTRUCTION OR DIRECTIVE WAS IN ERROR.
      IF(INSTER .EQ. .TRUE.) GO TO 40
      CALL ITMSCN
C
C OUTPUT INTERMEDIATE CODE TO LUN 11
40     CALL LINOUT
C
C END OF INPUT?
      IF(ENDFLG) GO TO 50
      GO TO 10
C
C BEGIN SECOND PASS
50     PASS = SECOND
      PC = 256
      ENDFLG = .FALSE.
      REWIND 11
      IF(LSTFLG) WRITE(13,100)
100    FORMAT(' LINE NO.  ADDRESS  LINE STATUS  OBJ CODE  LABEL
1 INSTR.  OP FIELD')
C
C REINITIALIZE LINE FLAGS
60     TYPERR = .FALSE.
      OPERR = .FALSE.
      LABERR = .FALSE.
      DIRERR = .FALSE.
      INSTER = .FALSE.
      RESOLV = .TRUE.

```

```

        WRDNUM = 1
        BYTNUM = 1
        LNSTAT = 0
        DIRNUM = 0
C
C SAVE PC FOR PROGRAM LISTING
        THISPC = PC
C
C CLEAR ASCII BUFFERS AND GIRL STRINGS
        DO 120 I1=1,40
120      BUF(I1) = BLANK
C
G        STRING=(LABELS,OPCOD)
G        ITEM  =(ITMTYP,VALUE,BUFPOS,LENID)
C
C READ IN THE INTERMEDIATE CODE, AND CONVERT BACK TO GIRL STRINGS
        CALL LINEIN
        IF(COMNTS) GO TO 140
C
C CREATE TARGET WORDS, UNLESS INSTRUCTION OR DIRECTIVE ERROR
        IF(INSTER.EQ..TRUE.) GO TO 130
        CALL ITMSCN
C
C CHECK FOR ERRORS
130      CALL ERRROUT(THISPC)
C
C GIVE A SOURCE LISTING IF DESIRED
140      IF(LSTFLG) CALL SRCLST(THISPC)
C
C CONVERT TO BYTES AND PLACE ON LUN 12
        IF(.NOT. COMNTS) CALL ABSOUT
C
C END OF PROGRAM ?
        IF(.NOT. ENDFLG) GO TO 60
        REWIND 14
        IF(LSTFLG) CALL VARLST
        STOP
C
G        COMPLETE
C
C
C
C
C        SUBROUTINE ADDREG
C
COMMON /ASSEMB/ PASS,REGSRC,REGDES,REGCOD(2),PC,LINEND,ADDRES,
1          MAJVAL,COMENT,IDSTAT,LNSTAT,ERRNUM,FMT,DIRNUM,
2          OPER1,OPER2,ER,ERLABL,WRDNUM,BYTNUM,ERROR
COMMON /GIRLCH/ EXCLAM,QUOTE,POUND,DOLLAR,PERCNT,AMPERS,APOST,
1          LPAR,RPAR,STAR,PLUS,COMMA,MINUS,PERIOD,SLASH,
2          ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,
3          NINE,COLON,SCOLON,LSTHAN,EQUALS,GTTHAN,QUEST,
4          ATSIGN,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,
5          V,W,X,Y,Z
COMMON /GIRL/  STRING,TOKEN,ITEM,BUFPOS,LENID,LABELS,OPCOD,STOP,
1          ITMTYP,VALUE,SYMBOL,REG,REGSTR,REGATS,REGPLS,
2          VARABL,NUMBER,LITRAL,LOGOR,LOGAND,LOGNOT,MODULO,
3          OFFIN,DIRECT,START,TOKODE
C
        DIMENSION NUMBUF(10)
C
        EQUIVALENCE(ZERO,NUMBUF(1))
C
C CREATE REGISTER NAME STRINGS R0, ..., R15 AND PLACE INTO NAME TAB.E
C TAGGED AS REGISTERS AND GIVEN ADDRESSES EQUAL TO THE REGISTER NUMBER.
C
C R0 - R9

```

```

      NODE = STRING
      IDENT = REG
      ADDRES = -1
G     STRING REG R
      DO 10 I1 = 1, 10
      ADDRES = ADDRES + 1
      NUM = NUMBUF(I1)
G     STRING REG - 2 NUM
      CALL ADDNAM(NODE, IDENT)
10    CONTINUE
C
C R10 - R15
G     STRING REG - 2 ONE
      DO 20 I1 = 1, 6
      NUM = NUMBUF(I1)
      ADDRES = ADDRES + 1
G     STRING REG - 3 NUM
      CALL ADDNAM(NODE, IDENT)
20    CONTINUE
G     COMPLETE
C
C
C
C
$
C     SUBROUTINE LEXSCN

      LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), HEXTBL, LINE
      LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
      LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
      LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1      DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG, HOL1, HOL2
C
      COMMON /ASC/      LINE(72), HEX(4), HEXCOD(4, 10)
      COMMON /ERRFLG/   LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
      COMMON /NAMFLG/   DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
      COMMON /LINFLG/   INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1      LSTFLG
      COMMON /ASSEMB/   PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1      MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2      OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
      COMMON /DATUM/    FIRST, SECOND, HEXTBL(16)
      COMMON /GIRLCH/   EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1      LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2      ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3      NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4      ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5      V, W, X, Y, Z
      COMMON /GIRL/     STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1      ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2      VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3      OPFIN, DIRECT, START, TOKODE
      COMMON /ASCII/    BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
C
      DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
      DIMENSION ASCGRL(58)
C
      EQUIVALENCE (LINE(1), LABEL(1))
      EQUIVALENCE (LINE(8), OPCODE(1))
      EQUIVALENCE (LINE(13), OPRBUF(1))
      EQUIVALENCE (ASCGRL(1), EXCLAM), (HEXCOD(1, 1), BUF(1))
C
C THIS ROUTINE IS A LEXICAL SCAN DESIGNED TO CREATE A BASIC TOKEN STRING
C CONSISTING OF OPERATORS, NUMBERS, ALPHANUMERICS, LITERALS, AND SPECIAL
C CHARACTERS. TWO ASSOCIATED STRINGS RELATE THE TOKEN STRING TO THE
C ORIGINAL ASCII STRING BY INDICATING EACH TOKEN'S STARTING POSITION IN
C ARRAY "OPRBUF" AND ALSO THAT TOKEN'S LENGTH.
C A LABEL STRING IS CREATED IN ORDER TO PLACE THAT LABEL INTO A NAME

```

```

C TABLE CALLED AN "EPAM" TREE.
C AN INSTRUCTION STRING IS CREATED IN ORDER TO SEARCH A PREVIOUSLY
C CREATED EPAM TREE WHICH DESCRIBES THE EXISTING INSTRUCTIONS AND
C DIRECTIVES.
C
C $-LITERAL: HOL1
      HOL1 = FALSE.
C '-LITERAL: HOL2
      HOL2 = FALSE.
C
C CHECK FOR LABEL AND CREATE LABEL STRING
      DO 10 I1=1,6
      ASCHAR = LABEL(I1)
      IF(ASCHAR .EQ. BLANK) GO TO 20
      GRCLHR = ASCGRL(ASCHAR-BLANK)
C
C IS THE ENTIRE LINE A COMMENT STATEMENT ?
      IF(GRCLHR .EQ. STAR) GO TO 15
G      STRING LABELS GRCLHR
10     CONTINUE
      GO TO 20
15     COMNTS = TRUE.
      RETURN
C
C CREATE INSTRUCTION STRING
20     DO 30 I1=1,4
      J1 = I1
      ASCHAR = OPCODE(J1)
      IF(ASCHAR .EQ. BLANK) GO TO 35
      GRCLHR = ASCGRL(ASCHAR-BLANK)
G      STRING OPCOD GRCLHR
30     CONTINUE
C
CHECK FOR EXTRANEIOUS CHARACTERS IN INSTRUCTION FIELD
35     IF(J1 .EQ. 4) GO TO 40
      J1 = J1 + 1
      ASCHAR = OPCODE(J1)
      IF(ASCHAR .NE. BLANK) INSTER = TRUE.
      GO TO 35
C
C CREATE TOKEN STRING
40     I1 = 0
      GRCLHR = 0
C
C ELIMINATE INITIAL BLANKS
45     I1 = I1 + 1
      IF(I1 .GT. 60) RETURN
      ASCHAR = OPRBUF(I1)
      IF(ASCHAR .EQ. BLANK) GO TO 45
C
C FIRST CHARACTER FOUND
      LENGTH = 0
      LSTSYM = GRCLHR
      GRCLHR = ASCGRL(ASCHAR-BLANK)
C
C IS IT A LITERAL?
      IF(GRCLHR .EQ. DOLLAR) GO TO 60
      IF(GRCLHR .EQ. APOST) GO TO 70
C
C IS IT A NUMBER?
      IF(LSTSYM .EQ. GTTHAN) GO TO 80
      IF((ASCHAR .GE. ASCII10) .AND. (ASCHAR .LE. ASCII19)) GO TO 80
C
C IS IT AN ALPHANUMERIC?
      IF((ASCHAR .GE. ASCII1A) .AND. (ASCHAR .LE. ASCII1Z)) GO TO 90
C

```

```

C CHARACTER IS EITHER AN OPERATOR OR A SPECIAL SYMBOL
  LENGTH = LENGTH + 1
  ISTLOC = I1
G   TOKEN STRING GRLCHR
Q55  TOKEN (BUFPOS "ISTLOC", LENID "LENGTH")
      IF(GRLCHR .NE. SCOLON) GO TO 45
C
C SEMI-COLON ENDS THE STATEMENT ALL FOLLOWING CHARACTERS ARE COMMENTS
  COMENT = I1
  RETURN
C
C $ LITERAL $
60   BOUND = DOLLAR
      GO TO 71
C
C ' LITERAL '
70   BOUND = APOST
Q71  TOKEN STRING LITRAL
      ISTLOC = I1 + 1
75   I1 = I1 + 1
      IF(I1 .GT. 60) GO TO 55
      ASCHAR = OPRBUF(I1)
      GRLCHR = ASCGRL(ASCHAR-BLANK)
      IF(GRLCHR .EQ. BOUND) GO TO 55
      LENGTH = LENGTH + 1
      GO TO 75
C
C NUMBER -- (HEX, DECIMAL, OR BINARY)
80   BOUND = ASCIIIF
G     TOKEN STRING NUMBER
      GO TO 91
C
C ALPHANUMERIC -- (IDENTIFIER OR REGISTER)
90   BOUND = ASCIIIZ
G     TOKEN STRING VARABL
91   ISTLOC = I1
95   I1 = I1 + 1
      LENGTH = LENGTH + 1
      ASCHAR = OPRBUF(I1)
      IF((ASCHAR .GE. ASCII10) .AND. (ASCHAR .LE. ASCII19)) GO TO 95
      IF((ASCHAR .GE. ASCII1A) .AND. (ASCHAR .LE. BOUND)) GO TO 95
      I1 = I1 - 1
      GO TO 55
C
G     COMPLETE
C
C
C
C
*     SUBROUTINE ADDNAM(NODE, IDENT)
C
  LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
  LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
  1        DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
C
  COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
  COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
  COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
  1        LSTFLG
  COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
  1        MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
  2        OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
  COMMON /GIRL/   STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
  1        ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
  2        VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
  3        OPFIN, DIRECT, START, TOKODE
  COMMON /ASCII/  BLANK, ASCII10, ASCII19, ASCII1A, ASCII1F, ASCII1Z

```

```

C
      DIMENSION NAME(6)
C
C THIS ROUTINE SEARCHES THE NAME TABLE FOR THE REQUESTED IDENTIFIER
C AND ADDS IT TO THE TABLE IF NOT FOUND AN ADDRESS IS RETURNED UNLESS
C THE IDENTIFIER IS NOT YET DEFINED (JUMP AHEAD).
C
      DO 10 I1 = 1,6
10      NAME(I1) = 0
      IF(IDENT .NE. REG) ADDRES = 0
      ER = SYMBOL
      IDSTAT = 0
      POS = 0
C
C EXAMINE INPUT STRING
G20      NODE + IDENT."POS=POS+1"/30 'GRLCHR
      NAME(POS) = GRLCHR
G        ER + GRLCHR/65 'ER/20
C
C IF FAILURE, IDENTIFIER IS A SUBSET OF AN EXISTING NAME
C (EG. "VAR" > "VAR1")
G30      ER + STOP(/40 'ADDRES, . 2 'IDSTAT)
C
C NAME FOUND
      IF(IDENT .NE. LABELS) RETURN
      ERLABL = ER
C
C TEST FOR MULTIPLE DEFINITIONS
      IF((IDSTAT .AND. DEFFLG) .EQ. 0) GO TO 40
C
C USE MOST RECENT DEFINITION
      ERROR = ERROR + 1
      IDSTAT = IDSTAT .OR. MLTFLG
      LABERR = .TRUE.
C
C UPDATE ADDRESS AND IDENTIFIER STATUS
40      IF(IDENT .EQ. REG) IDSTAT = IDSTAT .OR. REGFLG .OR. DEFFLG
      IF(IDENT .NE. LABELS) GO TO 45
      ADDRES = PC
      ERLABL = ER
      IDSTAT = IDSTAT .OR. DEFFLG
G45      ER STOP(- 1 "ADDRES", - 2 "IDSTAT")
C
C OUTPUT IDENTIFIER TO NAME FILE
      IF(.NOT. LABERR) WRITE(14) NAME
      RETURN
C
C NAME MUST BE ADDED TO THE TREE, ADD NEXT CHARACTER
G60      NODE + IDENT."POS=POS+1"/40 'GRLCHR
      NAME(POS) = GRLCHR
G65      ER GRLCHR $'ER
      GO TO 60
C
C COMPLETE
C
C
C
$      SUBROUTINE INSTRU
C
      LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
C
      COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
      COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1          MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2          OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
      COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
1          ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,

```

```

2          VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3          OPFIN, DIRECT, START, TOKODE

C
DIRNUM = 0
FMT = 0
I1 = 0
S1 = START
Q10 STRING+ OPCODE, "I1=I1+1" /20 'GRLCHR
D TYPE 15, S1, GRLCHR
D15 FORMAT(' S1, GRLCHR', 2X, I4, 2X, I4)
G S1 + GRLCHR 'S1/40/10
C
C INSTRUCTION NAME OR DIRECTIVE MAY HAVE BEEN FOUND IN TREE
C TEST FOR INSTRUCTION
Q20 S1+OPFIN (1/30 'FMT, 2'OPER1, 3'OPER2//RETURN)
OPER2 = 0
RETURN

C
C TEST FOR DIRECTIVE
Q30 S1 + DIRECT 'DIRNUM//RETURN
C
C FAILURE
40 INSTER = .TRUE.
ERROR = ERROR + 1
G COMPLETE
C
C
C
$ SUBROUTINE TOKSCN
C
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), HEXTBL, LINE, NEG
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

C
COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4 ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5 V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2 VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z

C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)

C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))

C
DATA DEC, BIN, HEXA /1, 2, 3/

C
C THIS ROUTINE

```

```

C 1) COMBINES (REDUCES) THE TOKENS FROM LEXSCN INTO "ITEMS"
C SUCH AS REGISTER TYPES, LOGICAL AND MATHEMATICAL OPERATORS,
C VARIABLES, NUMBERS, LITERALS, COMMAS, SEMI-COLONS,
C AND EXCLAMATION MARKS.
C 2) ASSIGNS: REGISTERS > REGISTER NUMBERS
C OPERATORS > 0
C VARIABLES > ADDRESS VALUE OR 0 IF UNDEFINED
C NUMBERS > DECIMAL VALUE
C LITERALS > FIRST TWO CHARACTERS
C COMMAS > 0
C SEMICOLON > 0
C EXCLAM MK > 0
C
C -----
C
C ITEM TYPE CODES: 0-18
C TOKEN CODES 8-22
C
C 0 REG 12 ,
C 1 *REG 13 !
C 2 @REG 14 ;
C 3 *REG+ 15 LITERAL
C 4 + LOG OR 16 NUMBER
C 5 * LOG AND 17 IDENTIFIER
C 6 - LOG NOT 18 @
C 7 / MODULUS 19 %
C 8 + 20 >
C 9 * 21 (
C 10 - 22 .
C 11 /
C
C -----
C
C INITIALIZE REGISTER CODES AND SOURCE AND DESTINATION REGISTER VALUES
C REGISTER CODES ARE 0 - 3
C
C REGCOD(1) = 0
C REGCOD(2) = 0
C REGSRC = 0
C REGDES = 0
C
C EXTRACT TOKEN
C I1 = 0
C COMMAS = 0
C
C DEFAULT: ASSUME NUMBERS ARE DECIMAL
C NUMTYP = DEC
C
C EXAMINE TOKEN STRING FOR FIRST CHARACTER OF ITEM
G10 TOKEN + STRING "I1=I1+1" /RETURN 'TOKE
G TOKEN +(BUFPOS I1 'ISTLOC, LENID I1 'LENGTH)
C
C OPERATOR AND DELIMITER VALUES DEFAULT TO 0 IN ORDER TO PAD
C ITEM - VALUE LIST
C VAL = 0
C
C BEGIN ITEM TYPE AND VALUE DEFINITIONS
G TOKODE + TOKE 'ITMVAL
C INDEX = ITMVAL - 7
C
C
C + * - / , ! , LIT NUM VAR
GO TO (2000, 100, 2000, 2000, 200, 2000, 2000, 300, 400, 500,
1 600, 700, 800, 900, 1000) INDEX
C @ % > (
C
C

```



```

C **** ASTERISK -- (STAR)
C     LOOK AHEAD, IF NOT A REGISTER, TREAT AS ARITHMETIC OPERATOR
C
100     TEMP1 = I1 + 1
G       TOKEN+(STRING,TEMP1'NEXT, BUFPOS,TEMP1'NXTLOC, LENID,TEMP1'NXTLEN)
        IF(NEXT .NE. VARABL) GO TO 2000
C
C NEXT TOKEN IS EITHER A REGISTER OR IDENTIFIER
C PLACE ON TEMPORARY STRING
        CALL TMPSTR(NXTLOC,NXTLEN)
        CALL ADDNAM(STRING,STRING)
        IF((IDSTAT .AND. REGFLG) .EQ. 0) GO TO 2000
C
C VARIABLE IS A REGISTER
        LENGTH = LENGTH + NXTLEN
        VAL = ADDRES
        I1 = I1 + 1
C
C ASSIGN SOURCE OR DESTINATION REGISTER VALUE
        IF(COMMAS .GE. 1) GO TO 120
        REGSRC = ADDRES
        GO TO 130
C
120     REGDES = ADDRES
C
C REGISTER MAY HAVE THIS FORM: *REG+
130     TEMP2 = TEMP1 + 1
G       TOKEN + STRING,TEMP2 'LAST
C
C SET REGISTER CODE
        ITMVAL = 1
        IF(LAST .NE. PLUS) GO TO 140
C *R+
        ITMVAL = 3
        I1 = I1 + 1
        LENGTH = LENGTH + 1
140     REGCOD(COMMAS + 1) = ITMVAL
        GO TO 2000
C
C **** COMMA
C
200     COMMAS = COMMAS + 1
        GO TO 2000
C
C **** LITERAL -- PLACE UP TO FIRST TWO CHARACTERS INTO "VAL"
C
300     VAL = OPRBUF(ISTLOC)
        IF(LENGTH .GT. 1) VAL = LVLFSH(VAL,8) .OR. OPRBUF(ISTLOC + 1)
        GO TO 2000
C
C **** NUMBER -- PLACE DECIMAL VALUE INTO VAL
C
400     GO TO (410, 430, 450) NUMTYP
C
C DECIMAL
410     DO 420 K1 = 1, LENGTH
        M1 = K1 + ISTLOC - 1
        VAL = 10 * VAL + (OPRBUF(M1) - ASCII0)
420     CONTINUE
        GO TO 2000
C
C BINARY
430     DO 440 K1 = 1, LENGTH
        M1 = K1 + ISTLOC - 1
        VAL = 2 * VAL + (OPRBUF(M1) - ASCII0)
440     CONTINUE

```

```

        GO TO 2000
C
C HEXADECIMAL
C FIRST CHECK FOR VALUES GT 32767
450     NEG = .FALSE.
        IF((LENGTH .NE. 4) .OR. (OPRBUF(ISTLOC) .LE. ASCII0 + 7))
1       GO TO 453
        NEG = .TRUE.
        CALL TWOCMP(OPRBUF(ISTLOC))
453     DO 460 K1 = 1, LENGTH
        HEXCHR = OPRBUF(K1 + ISTLOC - 1)
        DO 455 L1 = 1, 16
        J1 = L1
        IF(HEXCHR .EQ. HEXTBL(J1)) GO TO 457
455     CONTINUE
457     DECHAR = J1 - 1
        VAL = 16 * VAL + DECHAR
460     CONTINUE
        IF(.NOT. NEG) GO TO 2000
        VAL = -VAL
        CALL TWOCMP(OPRBUF(ISTLOC))
        GO TO 2000
C
C **** IDENTIFIER -- COULD BE A REGISTER (CODE = 0)
C
C PLACE ON TEMPORARY STRING
500     CONTINUE
        CALL TMPSTR(ISTLOC,LENGTH)
        CALL ADDNAM(STRING,STRING)
        VAL = ADDRESS
C
C REGISTER?
        IF((IDSTAT .AND. REGFLG) .EQ. 0) GO TO 2000
        ITMVAL = 0
510     IF(COMMAS .GE. 1) GO TO 520
C
C SOURCE
        REGSRC = ADDRESS
        GO TO 530
C
C DESTINATION
520     REGDES = ADDRESS
C
C CHECK FOR RIGHT PARENTHESIS
530     TEMP = I1 + 1
G       TOKEN + STRING. TEMP/2000 = RPAR/2000
        I1 = TEMP
        GO TO 2000
C
C **** AT-SIGN -- @, NEXT ITEM COULD BE A NUMBER, REGISTER, OR VARIABLE
C
600     TEMP = I1 + 1
G       TOKEN+(STRING. TEMP 'NEXT, BUFPOS. TEMP 'NXTLOC, LENID. TEMP 'NXTLEN)
        REGCOD(COMMAS + 1) = 2
        IF(NEXT .NE. VARABL) GO TO 2000
C
C CHECK FOR REGISTER
        CALL TMPSTR(NXTLOC,NXTLEN)
        CALL ADDNAM(STRING,STRING)
C
C REGISTER?
        IF((IDSTAT .AND. REGFLG) .EQ. 0) GO TO 2000
        VAL = ADDRESS
        I1 = TEMP
        LENGTH = LENGTH + NXTLEN
        ITMVAL = 2

```



```

2          VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3          OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/  BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z

C
  DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)

C
  EQUIVALENCE (LINE(1), LABEL(1))
  EQUIVALENCE (LINE(8), OPCODE(1))
  EQUIVALENCE (LINE(13), OPRBUF(1))

C
  DATA EVEN, ODD, SOURCE, DEST, MASK4, MASK8 /0, 1, 0, 1, "17, "377/
  DATA REGIST, ALL, NUM, KOMMA, LITERL /0, -1, 16, 12, 15/

C
C THIS ROUTINE SCANS THE ITEM STRING LEFT TO RIGHT.
C A) FIRST PASS - THE NUMBER OF TARGET WORDS NEEDED FOR EACH INSTRUCTION
C                IS COUNTED AND THE PC UPDATED BY THAT AMOUNT SO THAT
C                LABELS MAY BE GIVEN CORRECT ADDRESS VALUES PRIOR TO THE
C                SECOND PASS.
C B) SECOND PASS- THE TARGET MACHINE CODE IS CREATED.
C
C UP TO TEN WORDS MAY BE CREATED BY A DIRECTIVE
C   FOR EXAMPLE: "DATA" WITH TEN OPERANDS OR "BYTE" WITH TWENTY OPERANDS
C UP TO THREE WORDS MAY BE CREATED BY AN INSTRUCTION.
C
C LEFT TO RIGHT ARITHMETIC WITH NO PARENTHESES IS ALLOWED
C
  BYTNUM = 1
  WRDNUM = 1
  IF(DIRNUM .NE. 0) GO TO 2000
  BYTNUM = 2
  PC = PC + 2
  GO TO (100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200) FMT

C
C **** FORMAT 1 S, D
C
100  DESTIN = ALL
C
C FORM FIRST HEX WORD
110  CALL FMT1(ADRCOD(1))
      IF(PASS .EQ. FIRST) GO TO 120
      TEMP1 = LVRTSH(OPER1, 8)
      TEMP2 = TEMP1
      HEXCOD(1, 1) = TEMP2
      TEMP1 = (OPER1 .AND. MASK8)
      TEMP2 = TEMP1 - ASCII0
      IF(TEMP2 .GT. 9) TEMP2 = TEMP1 - ASCIIA + 10
      HEXCOD(2, 1) = HEXTBL(ADRCOD(1) + TEMP2 + 1)
      HEXCOD(3, 1) = HEXTBL(ADRCOD(2) + 1)
      HEXCOD(4, 1) = HEXTBL(ADRCOD(3) + 1)

C
C COMPUTE SOURCE AND DESTINATION FIELDS
120  IF(FMT .EQ. 4) RETURN
      I1 = 0
      FIELD = SOURCE
125  CALL OPFLD(I1)
C
C DESTINATION FIELD IS RESTRICTED TO SIMPLE REGISTERS FOR FORMATS 3 AND 9
      IF(FIELD .EQ. SOURCE) GO TO 130
      IF(DESTIN .EQ. ALL) GO TO 130
      IF(DESTIN .NE. MAJVAL) OPERR = .TRUE.

C
C WILL EXTRA WORDS BE NEEDED BEYOND THE BASIC INSTRUCTION WORD?
130  IF(.NOT. EXTRA) GO TO 150
      WRDNUM = WRDNUM + 1
      BYTNUM = BYTNUM + 2
      PC = PC + 2

```

```

        IF(PASS .EQ. FIRST) GO TO 150
        DO 140 J1 = 1,4
140      HEXCOD(J1,WRDNUM) = HEX(J1)
150      IF(FIELD .EQ. DEST) RETURN
        FIELD = DEST
        GO TO 125
C
C **** FORMAT 2 DISPLACEMENT -128 >= DISP >= 127 (JUMP STATEMENTS)
C
200      THSLOC = PC - 2
        IF(PASS .EQ. FIRST) RETURN
C
C COMPUTE ABSOLUTE ADDRESS
        I1 = 0
        CALL OPFLD(I1)
C
C COMPUTE RELATIVE DISPLACEMENT IN WORDS
        DISP = ((ADDRES - THSLOC) - 2) / 2
C
C IF DISPLACEMENT EXCEEDS LIMIT, SET ERROR FLAG
C AND USE DISPLACEMENT MODULO(128)
        IF((DISP .GT. 127) .OR. (DISP .LT. -128)) OPERR = .TRUE
C
C CONVERT DISPLACEMENT TO HEX
        CALL DECHEX(DISP,HEX(1))
        TEMP1 = LVRTSH(OPER1,8)
        TEMP2 = TEMP1
        HEXCOD(1,1) = TEMP2
        TEMP1 = (OPER1 AND MASK8)
        TEMP2 = TEMP1
        HEXCOD(2,1) = TEMP2
        HEXCOD(3,1) = HEX(3)
        HEXCOD(4,1) = HEX(4)
        RETURN
C
C **** FORMAT 3 S.W DESTINATION MUST BE A SIMPLE WORKSPACE REGISTER
C
300      DESTIN = REGIST
        GO TO 110
C
C **** FORMAT 4 S.C DESTINATION MUST BE AN INTEGER FROM 0 TO 15
C
400      I1 = 0
        FIELD = SOURCE
        CALL OPFLD(I1)
425      IF(FIELD .EQ. SOURCE) GO TO 430
C
C CHECK DESTINATION AGAINST 0-15 VALUE INTEGER
        IF(MAJVAL NE NUM) OPERR = .TRUE
        IF((ADDRES .LT. 0) OR (ADDRES .GT. 15)) OPERR = .TRUE
        REGDES = ADDRES
        GO TO 450
C
C EXTRA TARGET WORD NEEDED?
430      IF( NOT EXTRA) GO TO 450
        BYTNUM = BYTNUM + 2
        WRDNUM = WRDNUM + 1
        PC = PC + 2
        IF(PASS .EQ. FIRST) GO TO 450
        DO 440 J1 = 1,4
440      HEXCOD(J1,WRDNUM) = HEX(J1)
450      IF(FIELD .EQ. DEST) GO TO 110
        FIELD = DEST
        GO TO 425
C
C **** FORMAT 5 W.N

```

```

C
C SOURCE FIELD MUST CONTAIN A SIMPLE WORKSPACE REGISTER
C DESTINATION FIELD MUST CONTAIN AN INTEGER FROM 0 TO 15
500   IF(PASS .EQ. FIRST) RETURN
      I1 = 0
      CALL OPFLD(I1)

C
C SOURCE FIELD -- W
      IF(MAJVAL .NE. REGIST) OPERR = .TRUE
      TEMP1 = LVRTSH(OPER1,8)
      TEMP2 = TEMP1
      HEXCOD(1,1) = TEMP2
      TEMP1 = (OPER1 .AND. MASK8)
      TEMP2 = TEMP1
      HEXCOD(2,1) = TEMP2

C
      HEXCOD(4,1) = HEX(4)

C
C DESTINATION FIELD --      0 <= N <= 15
      CALL OPFLD(I1)
      IF(MAJVAL .NE. NUM) OPERR = .TRUE.
      HEXCOD(3,1) = HEX(4)
      RETURN

C
C **** FORMAT 6  S
C
600   I1 = 0
      CALL OPFLD(I1)
      IF(EXTRA) PC = PC + 2
      IF(PASS .EQ. FIRST) RETURN

C
      TEMP1 = LVRTSH(OPER1,8)
      TEMP2 = TEMP1
      HEXCOD(1,1) = TEMP2
      TEMP1 = (OPER1 .AND. MASK8)
      TEMP2 = TEMP1
      HEXCOD(2,1) = TEMP2
      TEMP1 = LVRTSH(OPER2,8)
      IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
      TEMP2 = TEMP1 - ASCII0
      IF(TEMP2 .GT. 9) TEMP2 = TEMP1 - ASCIIA + 10
      HEXCOD(3,1) = HEXTAB(HEXCOD(1) + TEMP2 + 1)

C
      IF(EXTRA) GO TO 610

C
C NO EXTRA WORD IS NEEDED FOR A SIMPLE REGISTER
      HEXCOD(4,1) = HEX(4)
      RETURN

C
C EXTRA WORD FOR LITERAL, NUMBER, OR MEMORY LOCATION
610   HEXCOD(4,1) = HEXTAB(REGSRC + 1)
      BYTNUM = BYTNUM + 2
      WRDNUM = WRDNUM + 1
      DO 620 J1 = 1,4
620   HEXCOD(J1,WRDNUM) = HEX(J1)
      RETURN

C
C **** FORMAT 7  N/A
C
700   IF(PASS .EQ. FIRST) RETURN
      TEMP1 = LVRTSH(OPER1,8)
      TEMP2 = TEMP1
      HEXCOD(1,1) = TEMP2
      TEMP1 = (OPER1 .AND. MASK8)
      TEMP2 = TEMP1
      HEXCOD(2,1) = TEMP2

```

```

        TEMP1 = LVRTSH(OPER2,8)
        IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
        TEMP2 = TEMP1
        HEXCOD(3,1) = TEMP2
        TEMP1 = (OPER2 AND MASK8)
        IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
        TEMP2 = TEMP1
        HEXCOD(4,1) = TEMP2
        RETURN
C
C ***** FORMAT 8 W.IOP
C
800      PC = PC + 2
        IF(PASS .EQ. FIRST) RETURN
C
C SOURCE MUST BE A SIMPLE REGISTER
C CONSTRUCT FIRST WORD
        I1 = 0
        CALL OPFLD(I1)
        IF(MAJVAL .NE. REGIST) OPERR = .TRUE.
        TEMP1 = LVRTSH(OPER1,8)
        TEMP2 = TEMP1
        HEXCOD(1,1) = TEMP2
        TEMP1 = (OPER1 AND MASK8)
        TEMP2 = TEMP1
        HEXCOD(2,1) = TEMP2
        TEMP1 = LVRTSH(OPER2,8)
        IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
        TEMP2 = TEMP1
        HEXCOD(3,1) = TEMP2
        HEXCOD(4,1) = HEX(4)
C
C CONSTRUCT IMMEDIATE OPERAND DO NOT RESET ITEM STRING POINTER I1
        CALL OPFLD(I1)
        BYTNUM = BYTNUM + 2
        WRDNUM = WRDNUM + 1
        DO 810 J1 = 1,4
810      HEXCOD(J1,WRDNUM) = HEX(J1)
        RETURN
C
C ***** FORMAT 9 S.W DESTINATION MUST BE A SIMPLE WORKSPACE REGISTER
C
900      DESTIN = REGIST
        GO TO 110
C
C ***** FORMAT A BIT # -- MUST BE A POSITIVE INTEGER, VALUE IS
C                                     COMPUTED MOD 256
1000     IF(PASS .EQ. 1) RETURN
        I1 = 0
        CALL OPFLD(I1)
        IF(MAJVAL .NE. NUM) OPERR = .TRUE.
        IF(ADDRES .LT. 0) OPERR = .TRUE.
        TEMP1 = LVRTSH(OPER1,8)
        TEMP2 = TEMP1
        HEXCOD(1,1) = TEMP2
        TEMP1 = (OPER1 AND MASK8)
        TEMP2 = TEMP1
        HEXCOD(2,1) = TEMP2
        HEXCOD(3,1) = HEX(3)
        HEXCOD(4,1) = HEX(4)
        RETURN
C
C ***** FORMAT B IOP
C
1100     PC = PC + 2
        IF(PASS .EQ. FIRST) RETURN

```

```

TEMP1 = LVRTSH(OPER1,8)
TEMP2 = TEMP1
HEXCOD(1,1) = TEMP2
TEMP1 = (OPER1 .AND. MASK8)
TEMP2 = TEMP1
HEXCOD(2,1) = TEMP2
TEMP1 = LVRTSH(OPER2,8)
IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
TEMP2 = TEMP1
HEXCOD(3,1) = TEMP2
TEMP1 = (OPER2 .AND. MASK8)
IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
TEMP2 = TEMP1
HEXCOD(4,1) = TEMP2
C
C FORM IOP
    I1 = 0
    CALL OPFLD(I1)
    BYTNUM = BYTNUM + 2
    WRDNUM = WRDNUM + 1
    DO 1110 J1 = 1,4
1110    HEXCOD(J1,WRDNUM) = HEX(J1)
    RETURN
C
C **** FORMAT C W -- OPERAND FIELD IS RESTRICTED TO SIMPLE REGISTERS
C
1200    IF(PASS .EQ. FIRST) RETURN
    I1 = 0
    CALL OPFLD(I1)
    IF(MAJVAL .NE. REGIST) OPERR = .TRUE.
    TEMP1 = LVRTSH(OPER1,8)
    TEMP2 = TEMP1
    HEXCOD(1,1) = TEMP2
    TEMP1 = (OPER1 .AND. MASK8)
    TEMP2 = TEMP1
    HEXCOD(2,1) = TEMP2
    TEMP1 = LVRTSH(OPER2,8)
    IF(TEMP1 .EQ. 0) TEMP1 = ASCII0
    TEMP2 = TEMP1
    HEXCOD(3,1) = TEMP2
    HEXCOD(4,1) = HEX(4)
    RETURN
C
C
C **** DIRECTIVES ****
C
D2000    TYPE 2001, DIRNUM
D2001    FORMAT(' DIRNUM = ', I5)
2000     GO TO(2100, 2200, 2300, 2400, 2500, 2600, 2700, 2800, 2900, 3000, 3100,
1         3200, 3300, 3400, 3500, 3600, 3700, 3800, 3900, 4000) DIRNUM
C
C **** ADRG
C
2100     COMNTS = .TRUE.
        GO TO 4000
        RETURN
C
C **** BES
C
2200     TYPE 2201
2201     FORMAT(' BES', *)
        TYPE 2205
2205     FORMAT('+', ' IS NOT YET IMPLEMENTED')
        COMNTS = .TRUE.
        RETURN

```



```

C
C **** BSS -- BLOCK STARTING WITH SYMBOL
C
2300      I1 = 0
          CALL OPFLD(I1)
          PC = PC + ADDRES
          RETURN

C
C **** BYTE -- INITIALIZE BYTE DATA, MUST END WITH SEMI-COLON
C
2400      NUMEXP = 1
          WRDPOS = EVEN
          LIMIT = 20
C          IF((PC .AND. ODD) .EQ. 0) GO TO 2405
C          LIMIT = 19
C          WRDPOS = ODD
2405      PC = PC + 1
C
C COUNT THE COMMAS
          I1 = 0
02410     ITEM + ITMTYP."I1 = I1 + 1"/2430 =KOMMA/2410/2420
2420      PC = PC + 1
          NUMEXP = NUMEXP + 1
          GO TO 2410
C
C IF THIS DIRECTIVE HAD A LABEL, DECLARE IT AS BYTE
02430     STRING + LABELS/2450
C          ER + STOP.2 'IDSTAT
          IDSTAT = IDSTAT .OR. BYTFLG
C          ER STOP -.2 "IDSTAT"
C
C BEGIN EVALUATION OF THE EXPRESSIONS; 20 LIMIT IF EVEN, 19 IF ODD
02450     LIMIT = 20
C          IF(WRDPOS .EQ. ODD) LIMIT = 19
2450      BYTNUM = NUMEXP
          WRDNUM = 1
          I1 = 0
          IF(NUMEXP .GT. LIMIT) OPERR = .TRUE.
          DO 2470 J1 = 1, NUMEXP
          CALL OPFLD(I1)
          IF(WRDPOS .EQ. ODD) GO TO 2460
C PC IS EVEN
          HEXCOD(1, WRDNUM) = HEX(3)
          HEXCOD(2, WRDNUM) = HEX(4)
          WRDPOS = ODD
          GO TO 2470
C PC IS ODD
2460      HEXCOD(3, WRDNUM) = HEX(3)
          HEXCOD(4, WRDNUM) = HEX(4)
          WRDPOS = EVEN
          WRDNUM = WRDNUM + 1
2470      CONTINUE
          IF(WRDPOS .EQ. EVEN) WRDNUM = WRDNUM -1
          RETURN
C
C **** DATA -- INITIALIZE WORD DATA
C
2500      NUMEXP = 1
          PC = PC + 2
C
C COUNT THE COMMAS
          I1 = 0
02510     ITEM + ITMTYP."I1 = I1 + 1"/2530 =KOMMA/2510/2520
2520      PC = PC + 2
          NUMEXP = NUMEXP + 1
          GO TO 2510

```

```

C
C EVALUATE UP TO 10 EXPRESSIONS
2530  LIMIT = 10
      IF(NUMEXP GT LIMIT) OPERR = TRUE
      BYTNUM = 2*NUMEXP
      I1 = 0
      WRDNUM = 0
      DO 2570 J1 = 1, NUMEXP
        CALL OPFLD(I1)
        WRDNUM = WRDNUM + 1
        DO 2560 L1 = 1, 4
          HEXCOD(L1, WRDNUM) = HEX(L1)
2560
2570  CONTINUE
      RETURN
C
C **** DEF
C
2600  TYPE 2601
2601  FORMAT(' DEF ', $)
      TYPE 2205
      COMNTS = TRUE
      RETURN
C
C **** DORG
C
2700  TYPE 2701
2701  FORMAT(' DORG ', $)
      TYPE 2205
      COMNTS = TRUE
      RETURN
C
C **** DXOP -- DEFINE EXTENDED OPERATION
C
2800  COMNTS = TRUE
      IF(PASS EQ SECOND) RETURN
C
C REMOVE LABEL FROM IDENTIFIER TREE
G      ERLABL - STOP
C
C ADD LABEL TO "INSTRUCTION - DIRECTIVE" TREE
      S1 = START
      I1 = 0
G20    STRING + LABELS "I1 = I1 + 1"/30 'GRLCHR
G      S1 + GRLCHR 1 /25 'S1 /20
G25    S1 GRLCHR $'S1
      GO TO 20
C
C USE FORMAT "6"
G30    S1 OPFIN "6"
C
C DETERMINE EXTENDED OPERATION NUMBER AND INCORPORATE INTO
C XOP INSTRUCTION SKELETON
      I1 = 0
      CALL OPFLD(I1)
      IF(MAJVAL NE NUM) OPERR = TRUE
C
C SPLIT XOP NUMBER INTO TWO RIGHT AND TWO LEFT BITS
      RTHALF = ADDRES .AND 3
      LFHALF = LVRTSH(ADDRES, 2)
C
C CREATE INSTRUCTION SKELETON
      ISTBYT = HEXTBL(2 + 1)
      SECBYT = HEXTBL(12 + LFHALF + 1)
      ISTWRD = LVLF5H(ISTBYT, 8) OR SECBYT
      BYTONE = LVLF5H(RTHALF, 2)
      BYTONE = HEXTBL(BYTONE + 1)

```

```

        SECWRD = LVLFSH(BYTONE, 8)
G      S1 OPFIN ( _1STWRD, _SECWRD)
        RETURN
C
C **** END  --  END OF ASSEMBLY
C
2900    ENDFLG = .TRUE
        RETURN
C
C **** EQU  --  DEFINE ASSEMBLY TIME CONSTANT
C          SET LABEL TO VALUE IN OPERAND FIELD
3000    I1 = 0
        CALL OPFLD(I1)
G      ERLABL STOP - 1 "ADDRES"
        IF(PASS EQ SECOND) COMNTS = TRUE
        RETURN
C
C **** EVEN  --  IF PROGRAM COUNTER IS AN ODD NUMBER, ADD ONE TO EVEN UP
C
3100    IF((PC AND ODD) EQ 0) GO TO 3150
        PC = PC + 1
        HEXCOD(1, WRDNUM) = ASCII0
        HEXCOD(2, WRDNUM) = ASCII0
        RETURN
3150    COMNTS = .TRUE
        RETURN
C
C **** IDT
C
3200    TYPE 3201
3201    FORMAT(' IDT', $)
        TYPE 2205
        COMNTS = .TRUE
        RETURN
C
C **** LIST  --  OUTPUT SOURCE LISTING
C
3300    LSTFLG = .TRUE
        COMNTS = .TRUE
        RETURN
C
C **** PAGE
C
3400    TYPE 3401
3401    FORMAT(' PAGE', $)
        TYPE 2205
        COMNTS = .TRUE
        RETURN
C
C **** REF
C
3500    TYPE 3501
3501    FORMAT(' REF', $)
        TYPE 2205
        COMNTS = .TRUE
        RETURN
C
C **** RORG
C
3600    TYPE 3601
3601    FORMAT(' RORG', $)
        TYPE 2205
        COMNTS = .TRUE
        RETURN
C
C **** TEXT  --  INITIALIZE UP TO 20 CHARACTERS IF PC IS EVEN, 19 IF ODD

```

```

C
3700    LIMIT = 20
        WRDPOS = EVEN
        IF((PC .AND. ODD) .EQ. 0) GO TO 3710
        LIMIT = 19
        WRDPOS = ODD

C
C OBTAIN LOCATION AND LENGTH OF TEXT
@3710    ITEM + (ITMTYP 'ITMVAL, BUFPOS 'ISTLOC, LENID 'LENGTH)
C
C ERROR IF NOT LITERAL OR EXCEEDS 19 OR 20 CHARACTERS
        IF(ITMVAL .NE. LITERL) OPERR = .TRUE.
        IF(LENGTH .GT. LIMIT) OPERR = .TRUE.
        BYTNUM = LENGTH
        PC = PC + LENGTH
        IF(PASS .EQ. FIRST) RETURN
        WRDNUM = 1

C
C PICK UP NEXT CHARACTER
        NXTLOC = ISTLOC
        DO 3770 J1 = 1, LENGTH
            IF(WRDPOS .EQ. ODD) GO TO 3760
            TEMP2 = OPRBUF(NXTLOC)/16
            HEXCOD(1,WRDNUM) = HEXTBL(TEMP2 + 1)
            TEMP2 = OPRBUF(NXTLOC) .AND. MASK4
            HEXCOD(2,WRDNUM) = HEXTBL(TEMP2 + 1)
            NXTLOC = NXTLOC + 1
            WRDPOS = ODD
            GO TO 3770

C
3760    TEMP2 = OPRBUF(NXTLOC)/16
        HEXCOD(3,WRDNUM) = HEXTBL(TEMP2 + 1)
        TEMP2 = OPRBUF(NXTLOC) .AND. MASK4
        HEXCOD(4,WRDNUM) = HEXTBL(TEMP2 + 1)
        NXTLOC = NXTLOC + 1
        WRDPOS = EVEN
        WRDNUM = WRDNUM + 1

3770    CONTINUE
        RETURN

C
C **** TITL
C
3800    TYPE 3801
3801    FORMAT(' TITL',*)
        TYPE 2205
        COMNTS = .TRUE.
        RETURN

C
C **** UNL
C
3900    LSTFLG = .FALSE.
        COMNTS = .TRUE.
        RETURN

C
C **** SLASH -- CHANGE PROGRAM COUNTER
C
4000    I1 = 0
        CALL OPFLD(I1)
        PC = ADDRES
        RETURN

G      COMPLETE

```

```

C
C
C
*
C
SUBROUTINE LINOUT

LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, LINE
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG

C
COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINEND, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFP, S, LENID, LABELS, OPCOD, STOP,
1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2 VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OPFIN, DIRECT, START, TOKODE
COMMON /TMPARR/ TEMP1(30), TEMP2(30), TEMP3(30), TEMP4(30)

C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)

C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))

C
C *** OUTPUT INTERMEDIATE CODE (ITEM STRING) FOR A SINGLE LINE
WRITE(11) REGSRC, REGDES, REGCOD, LINEND,
1 COMENT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, LINE,
3 INSTER, COMNTS, LABERR, OPERR

C
C EXTRACT INTERMEDIATE CODE FROM THE ITEM STRING UNLESS COMMENT
IF(COMNTS EQ TRUE) RETURN
LIMIT = 30
I1 = 0
G10 ITEM +(ITMTYP, "I1=I1+1" /20 'ITMVAL, VALUE I1 'VAL)
G ITEM +(BUFP, I1 'ISTLOC, LENID I1 'LENGTH)
TEMP1(I1) = ITMVAL
TEMP2(I1) = VAL
TEMP3(I1) = ISTLOC
TEMP4(I1) = LENGTH
GO TO 10
20 I1 = I1 - 1
IF(I1 GT LIMIT) GO TO 99
WRITE(11) TEMP1, TEMP2, TEMP3, TEMP4
RETURN

C
C ERROR
99 TYPE 100
100 FORMAT(' *** ERROR - TOO MANY ITEMS IN OPERAND FIELD')
STOP
G COMPLETE

```

```

%      SUBROUTINE LINEIN
C
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, BUF(40), LINE
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1      DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG, DONE
C
COMMON /ASC/      LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/   LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/   DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
COMMON /LINFLG/   INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1      LSTFLG
COMMON /ASSEMB/   PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1      MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2      OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /GIRLCH/   EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1      LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2      ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3      NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4      ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5      V, W, X, Y, Z
COMMON /GIRL/     STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
1      ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2      VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3      OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/    BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
COMMON /TMPARR/   TEMP1(30), TEMP2(30), TEMP3(30), TEMP4(30)
C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
DIMENSION ASCGRL(58)
C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
EQUIVALENCE (ASCGRL(1), EXCLAM), (HEXCOD(1, 1), BUF(1))
C
C *** INPUT INTERMEDIATE CODE (ITEM STRING) FOR A SINGLE LINE
READ(11) REGSRC, REGDES, REGCOD, LINENO,
1      COMENT, LNSTAT, ERRNUM, FMT, DIRNUM,
2      OPER1, OPER2, ER, ERLABL, LINE,
3      INSTER, COMNTS, LABERR, OPERR
C
C IS THE ENTIRE LINE A COMMENT STATEMENT ?
IF(COMNTS .EQ. TRUE) RETURN
C
C RECREATE LABEL AND ITEM STRINGS FROM THE INTERMEDIATE CODE
DONE = .FALSE.
READ(11) TEMP1, TEMP2, TEMP3, TEMP4
DO 10 I1 = 1, 30
IF(DONE) GO TO 20
ITEM ITMTYP "TEMP1(I1)"
ITEM VALUE "TEMP2(I1)"
ITEM BUFPOS "TEMP3(I1)"
ITEM LENID "TEMP4(I1)"
C
C LOOK FOR SEMICOLON
IF(TEMP1(I1) .EQ. 14) DONE = .TRUE.
10 CONTINUE
C
C CHECK FOR LABEL AND CREATE LABEL STRING
20 DO 30 I1=1, 6
ASCHAR = LABEL(I1)
IF(ASCHAR .EQ. BLANK) RETURN
ORLCHR = ASCGRL(ASCHAR-BLANK)
STRING LABELS ORLCHR

```

```

30      CONTINUE
      RETURN
      COMPLETE
C
C
C
C
$      SUBROUTINE OPFLD(I1)
      LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, HEXTBL, LINE
      LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
      LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY, DUMB
      LOGICAL*1 DEFFLG, REGFLG, MLTFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
      1      DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
C
      COMMON /ASC/      LINE(72), HEX(4), HEXCOD(4, 10)
      COMMON /ERRFLG/   LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
      COMMON /NAMFLG/   DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB
      COMMON /LINFLG/   INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
      1      LSTFLG
      COMMON /ASSEMB/   PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
      1      MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
      2      OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
      COMMON /DATUM/    FIRST, SECOND, HEXTBL(16)
      COMMON /GIRLCH/   EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
      1      LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
      2      ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
      3      NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
      4      ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
      5      V, W, X, Y, Z
      COMMON /GIRL/     STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
      1      ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
      2      VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
      3      OFFIN, DIRECT, START, TOKODE
      COMMON /ASCII/    BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
C
      DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
C
      EQUIVALENCE (LINE(1), LABEL(1))
      EQUIVALENCE (LINE(8), OPCODE(1))
      EQUIVALENCE (LINE(13), OPRBUF(1))
C
      DIMENSION OPRAND(2)
C
C THE PURPOSE OF THIS ROUTINE IS TO EITHER RETURN THE DECIMAL VALUE OF AN
C OPERAND FIELD (FROM COMMA TO COMMA) OR REGISTER NUMBER IN "ADDRES" AND
C THE HEX VALUE IN "HEX()" AND TO DETERMINE IF AN EXTRA WORD IS NEEDED
C FOR THE TARGET CODE. IF SO, "EXTRA" = TRUE.
C
      MAJVAL = 0
      EXTRA = FALSE
      OPRAND(FIRST) = 0
      OPRAND(SECOND) = 0
      OPRAN = FIRST
C
C
C PICK UP ALL ITEMS UNTIL COMMA OR SEMI-COLON
G10      ITEM + ITMTYP "I1 = I1 + 1"/30 'ITMVAL
G        ITEM + VALUE I1 'ADDRES
C
C IS THIS ITEM A REGISTER?
      IF(ITMVAL GT 3) GO TO 15
C
C CHECK FOR @VAR(REG) OR @NUM(REG)
      IF(MAJVAL EQ 17) GO TO 10
      IF(MAJVAL EQ 16) GO TO 10
      MAJVAL = ITMVAL
      OPRAND(OPRAN) = ADDRES
      GO TO 10

```

```

C
15      ITM = ITMVAL - 3
        IF(ITM GT 8) GO TO 20
C
C ITEM IS AN OPERATOR
        OPRATR = ITM
C
C SWITCH TO SECOND OPERAND   __ OP ' __'
        OPRAN = SECOND
        GO TO 10
C
C ITEM IS COMMA, EXCLAM MK, SEMI-COLON, LITERAL, NUMBER, IDENTIFIER, OR @
20      ITM = ITM - 8
        IF((ITM LT 1) OR (ITM GT 7)) GO TO 100
C
C          ,   '   ;   LIT   NUM   VAR   @
C          GO TO (30, 40, 30, 50, 50, 60, 70) ITM
C
C *** COMMA OR SEMI-COLON
C
C FIELD HAS BEEN COMPLETELY EXAMINED. NO EXTRA WORDS ARE NEEDED IF FIELD
C CONTAINED AN "UNMODIFIED" REGISTER
30      ADDRES = OPRAND(FIRST)
C
C CONVERT OPERAND FROM DECIMAL TO HEXADECIMAL
        CALL DECHEX(ADDRES, HEX(1))
        RETURN
C
C *** EXCLAMATION MARK
C
40      OPRAND(OPRAN) = PC
        GO TO 55
C
C *** LITERAL OR NUMBER
C
50      OPRAND(OPRAN) = ADDRES
55      EXTRA = .TRUE
        MAJVAL = ITMVAL
        IF(OPRAN EQ FIRST) GO TO 10
C
C AN OPERAND - OPERATOR - OPERAND TRIPLE HAS BEEN FOUND.
C COMPUTE IT AND RETURN RESULT INTO OPERAND NO. 1
        CALL COMPUT(OPRAND(1), OPRATR)
        GO TO 7
C
C *** IDENTIFIER (EXCLUDING REGISTER NAMES)
C
C WAS THE IDENTIFIER DEFINED WHEN THIS STATEMENT OCCURED?
60      IF(ADDRES GT 0) GO TO 50
C
C OBTAIN ADDRESS
G      ITEM + BUFPOS 11 'ISTLOC
G      ITEM + LENID 11 'LENGTH
        CALL TMPSTR(ISTLOC, LENGTH)
        CALL ADDNAM(STRING, STRING)
        IF(PASS EQ FIRST) GO TO 50
C
C STILL UNDEFINED?
        IF(ADDRES EQ 0) OPERR = .TRUE
        GO TO 50
C
C *** AT-SIGN @ -- NEXT ITEM IS A NUMBER OR VARIABLE
C
C ANTICIPATE SHORTHAND "@(REG)" FOR "@0(REG)"
70      EXTRA = TRUE
        ADDRES = 0

```



```

      MAJVAL = 16
      GO TO 10
100  OPERR = TRUE
      COMPLETE

      SUBROUTINE TMPSTR(ISTLOC,LENGTH)

      LOGICAL*1 LABEL,OPCODE,OPRBUF,HEX,HEXCOD,BUF(40),LINE
      LOGICAL*1 BLANK,ASCII0,ASCII9,ASCIIA,ASCIIIF,ASCIIIZ,ASCHAR

      COMMON /ASC/      LINE(72),HEX(4),HEXCOD(4,10)
      COMMON /ASSEMB/    PASS,REGSRC,REGDES,REGCOD(2),PC,LINENO,ADDRES,
1      MAJVAL,COMENT,IDSTAT,LNSTAT,ERRNUM,FMT,DIRNUM,
2      OPER1,OPER2,ER,ERLABL,WRDNUM,BYTNUM,ERROR
      COMMON /GIRLCH/    EXCLAM,QUOTE,POUND,DOLLAR,PERCNT,AMPERS,APOST,
1      LPAR,RPAR,STAR,PLUS,COMMA,MINUS,PERIOD,SLASH,
2      ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,
3      NINE,COLON,SCOLON,LSTHAN,EQUALS,GTTHAN,QUEST,
4      ATSIGN,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,
5      V,W,X,Y,Z
      COMMON /GIRL/      STRING,TOKEN,ITEM,BUFPOS,LENID,LABELS,OPCOD,STOP,
1      ITMTYP,VALUE,SYMBOL,REG,REGSTR,REGATS,REGPLS,
2      VARABL,NUMBER,LITRAL,LOGOR,LOGAND,LOGNOT,MODULO,
3      OPFIN,DIRECT,START,TOKODE
      COMMON /ASCII/     BLANK,ASCII0,ASCII9,ASCIIA,ASCIIIF,ASCIIIZ

      DIMENSION LABEL(6),OPCODE(4),OPRBUF(60)
      DIMENSION ASCGRL(58)

      EQUIVALENCE (LINE(1),LABEL(1))
      EQUIVALENCE (LINE(8),OPCODE(1))
      EQUIVALENCE (LINE(13),OPRBUF(1))
      EQUIVALENCE (ASCGRL(1),EXCLAM), (HEXCOD(1,1),BUF(1))

      C
      G      STRING - STRING
      LOC = ISTLOC
      DO 10 K1 = 1, LENGTH
      ASCHAR = OPRBUF(LOC)
      GRLCHR = ASCGRL(ASCHAR - BLANK)
      G      STRING STRING GRLCHR
      LOC = LOC + 1
10     CONTINUE
      RETURN
      COMPLETE

      SUBROUTINE VARLST

      LOGICAL*1 LABEL,OPCODE,OPRBUF,HEX,HEXCOD,LINE
      LOGICAL*1 BLANK,ASCII0,ASCII9,ASCIIA,ASCIIIF,ASCIIIZ,ASCHAR
      LOGICAL*1 DEFFLG,LABFLG,REGFLG,MLTFLG,BYTFLG,DUMB,EKS
      LOGICAL*1 IDNTRF(6),STATUS(6)

      C
      COMMON /ASC/      LINE(72),HEX(4),HEXCOD(4,10)
      COMMON /ASSEMB/    PASS,REGSRC,REGDES,REGCOD(2),PC,LINENO,ADDRES,
1      MAJVAL,COMENT,IDSTAT,LNSTAT,ERRNUM,FMT,DIRNUM,
2      OPER1,OPER2,ER,ERLABL,WRDNUM,BYTNUM,ERROR
      COMMON /GIRLCH/    EXCLAM,QUOTE,POUND,DOLLAR,PERCNT,AMPERS,APOST,
1      LPAR,RPAR,STAR,PLUS,COMMA,MINUS,PERIOD,SLASH,
2      ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,
3      NINE,COLON,SCOLON,LSTHAN,EQUALS,GTTHAN,QUEST,
4      ATSIGN,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,
5      V,W,X,Y,Z

```

```

COMMON /GIRL/  STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1              ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2              VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3              OFFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, MLTFLG, BYTFLG, DUMB

C
C      DIMENSION NAME(6), ALPHA(26), NUMBR5(10)
C
C      EQUIVALENCE (ALPHA(1), A), (NUMBR5(1), ZERO)
C
C      DATA EKS /1HX/
C
C      THIS ROUTINE OUTPUTS THE STATUS FOR EACH IDENTIFIER AND REGISTER
C
C      WRITE(13,1)
1      FORMAT(///, ' **** IDENTIFIER LISTING ****', //,
2      1 ' NAME          STATUS', //,
3      2 12X, 'ADDRESS   REG UNDEF MULT DEF  BYTE', //)
C
C      READ(14, END = 200) NAME
C      SY = SYMBOL
C
C      DO 10 J1 = 1, 6
C      IDNTR(J1) = BLANK
10     STATUS(J1) = BLANK
C
C      DO 100 I1 = 1, 6
C      GR1CHR = NAME(I1)
C      IF (GR1CHR .EQ. 0) GO TO 120
C
C      OBTAIN ASCII CHAR FROM GIRL VALUE
C      DO 20 K1 = 1, 26
C      L1 = K1
20     IF (GR1CHR .EQ. ALPHA(L1)) GO TO 50
C
C      DO 30 K1 = 1, 10
C      L1 = K1
30     IF (GR1CHR .EQ. NUMBR5(L1)) GO TO 40
40     IDNTR(I1) = ASCII0 + L1 - 1
C      GO TO 60
50     IDNTR(I1) = ASCII1A + L1 - 1
060    SY + GR1CHR 'SY
100    CONTINUE
C
0120   SY + STOP (/5 'ADDRES, 2 'IDSTAT)
      REGTST = IDSTAT .AND. REGFLG
      DEFTST = IDSTAT .AND. DEFFLG
      MLTTST = IDSTAT .AND. MLTFLG
      BYTTST = IDSTAT .AND. BYTFLG
      IF (REGTST .GT. 0) STATUS(1) = EKS
      IF (DEFTST .EQ. 0) STATUS(2) = EKS
      IF (MLTTST .GT. 0) STATUS(3) = EKS
      IF (BYTTST .GT. 0) STATUS(4) = EKS
C
C      CALL DECHEX (ADDRES, HEX)
C      WRITE(13, 150) IDNTR, ADDRES, HEX, STATUS
150    FORMAT(1X, 6A1, 1X, I6, 1X, 4A1, 4X, A1, 5X, A1, 8X, A1, 6X, 3A1)
C      GO TO 5
200    RETURN
0      COMPLETE
/      COMPLETE

```

```

C
C
C
      SUBROUTINE COMPUT(OPRAND,OPRATR)
      IMPLICIT INTEGER (A-Z)
C
      DIMENSION OPRAND(2)
C
C THIS ROUTINE PERFORMS A COMPUTATION ON OPRAND(1) OPRATR OPRAND(2)
C AS DETERMINED BY THE ARITHMETIC OR LOGICAL OPERATOR IN "OPRATR" AND
C PLACES THE RESULT INTO OPRAND(1).
C
C
C      + * - / + * - /
      GO TO ( 10, 20, 30, 40, 50, 60, 70, 80) OPRATR
C
C LOGICAL OR
C
10      OPRAND(1) = OPRAND(1) .OR. OPRAND(2)
      RETURN
C
C LOGICAL AND
C
20      OPRAND(1) = OPRAND(1) .AND. OPRAND(2)
      RETURN
C
C LOGICAL NOT -- ONE'S COMPLEMENT
C
30      OPRAND(1) = .NOT OPRAND(2)
      RETURN
C
C MODULO
C
40      OPRAND(1) = MOD(OPRAND(1),OPRAND(2))
      RETURN
C
C ADD
C
50      OPRAND(1) = OPRAND(1) + OPRAND(2)
      RETURN
C
C MULTIPLY
C
60      OPRAND(1) = OPRAND(1) * OPRAND(2)
      RETURN
C
C SUBTRACT
C
70      OPRAND(1) = OPRAND(1) - OPRAND(2)
      RETURN
C
C DIVIDE
C
80      OPRAND(1) = OPRAND(1) / OPRAND(2)
      RETURN
      END
C
C
C

```

```

SUBROUTINE DECHEX(DECNUM, HEX)
IMPLICIT INTEGER (A-Z)

C
LOGICAL*1 HEX(4), HEXTBL, NEG
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ, ASCHAR
C
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCIIA, ASCIIF, ASCIIZ
C
C DIMENSION TEMP(4)
C
C THIS ROUTINE CONVERTS THE DECIMAL VALUE IN "DECNUM" TO HEXADECIMAL
C AND PLACES IT INTO HEX()
C
NEG = .FALSE.
IF(DECNUM .LT. 0) NEG = .TRUE.
NUMBER = IABS(DECNUM)
DO 5 L1 = 1, 4
TEMP(L1) = 0
5 HEX(L1) = ASCII0
HEXPOS = 5
10 HEXPOS = HEXPOS - 1
C
C COMPUTE REMAINDER FROM MODULUS 16
REM = MOD(NUMBER, 16)
TEMP(HEXPOS) = REM
HEX(HEXPOS) = HEXTBL(REM + 1)
IF(NUMBER .LT. 16) GO TO 20
NUMBER = NUMBER / 16
GO TO 10
C
C IF NEGATIVE, CONVERT TO TWO'S COMPLEMENT FORM
20 IF(.NOT. NEG) RETURN
CALL TWOCMP(HEX(1))
RETURN
END

C
C
C
SUBROUTINE FMT1(ADRCOD)
IMPLICIT INTEGER (A-Z)
C
LOGICAL*1 HEXTBL, ADRCOD(3)
C
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
C
C
C ADRCOD(1) = (REGCOD(2) * 4) + (REGDES/4)
C
C ADRCOD(2) = REGCOD(1) + (4 * MOD(REGDES, 4))
C
C ADRCOD(3) = REGSRC
RETURN
END

C
C
C
FUNCTION LVLSH(WORD, BITS)
IMPLICIT INTEGER(A-Z)
C
C THIS FUNCTION PERFORMS A LEFT LOGICAL SHIFT
C
IF(BITS EQ 0) GO TO 10
IF(BITS GE 16) GO TO 20

```

```

      LVLFSH = WORD * 2 ** (BITS)
      RETURN
10    LVLFSH = WORD
      RETURN
20    LVLFSH = 0
      RETURN
      END

C
C
C
      FUNCTION LVRTSH(WORD,BITS)
      IMPLICIT INTEGER(A-Z)

C
C THIS FUNCTION PERFORMS A RIGHT LOGICAL SHIFT
C
      IF(BITS EQ 0) GO TO 10
      IF(BITS GE 16) GO TO 20
      LVRTSH = WORD / 2 ** (BITS)
      RETURN
10    LVRTSH = WORD
      RETURN
20    LVRTSH = 0
      RETURN
      END

C
C
C
      SUBROUTINE ABSOUT
      IMPLICIT INTEGER (A-Z)

C
      LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, HEXTBL, LINE
      LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
      LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
      LOGICAL*1 DEFFLG, REGFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1      DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
      LOGICAL*1 HEXBYT(2,20), PGMCTR, DUMMY2, LIN

C
      COMMON /ASC/      LINE(72), HEX(4), HEXCOD(4,10)
      COMMON /ERRFLG/   LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
      COMMON /NAMFLG/   DEFFLG, LABFLG, REGFLG, BYTFLG
      COMMON /LINFLG/   INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1      LSTFLG
      COMMON /ASSEMB/   PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1      MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2      OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
      COMMON /DATUM/    FIRST, SECOND, HEXTBL(16)
      COMMON /GIRLCH/   EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APDST,
1      LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2      ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3      NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4      ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5      V, W, X, Y, Z
      COMMON /GIRL/     STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCOD, STOP,
1      ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2      VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3      OPFIN, DIRECT, START, TOKODE
      COMMON /ASCII/    BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
      COMMON /LOCAL/    NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),
1      DUMMY2, OLDP, OLDBYT, LIN(48)

C
C
      DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)

      EQUIVALENCE (LINE(1), LABEL(1))
      EQUIVALENCE (LINE(8), OPCODE(1))
      EQUIVALENCE (LINE(13), OPRBUF(1))
      EQUIVALENCE (HEXCOD(1,1), HEXBYT(1,1))

```

```

C
DATA PGMCTR /1H0,1H1,1H0,1H0,1H /
DATA NUM1,NUM2,NUM3,OLDPC,OLDBYT /0,1,0,256,0/
DATA LINCT1,LINCT2,LINCT3 /1,2,3/
C
C TEST FOR MODIFICATION OF PC
IF((DIRNUM EQ 3) OR (DIRNUM EQ 20)) GO TO 500
C
DO 180 I1 = 1, BYTNUM
LIN(LINCT1) = BLANK
LIN(LINCT2) = HEXBYT(1,I1)
LIN(LINCT3) = HEXBYT(2,I1)
C
C HAS A LINE BEEN FILLED?
IF(LINCT3 GE (16*3)) GO TO 120
C
C IF NOT FILLED, IS THIS THE LAST LINE OF THE PROGRAM?
IF(ENDFLG) GO TO 200
LINCT1 = LINCT1 + 3
LINCT2 = LINCT2 + 3
LINCT3 = LINCT3 + 3
GO TO 180
C
C OUTPUT THIS LINE
120 WRITE(12,101) PGMCTR,LIN
101 FORMAT(53A1)
C
C UPDATE COUNTERS AND THE PROGRAM COUNTER ARRAY
LINCT1 = 1
LINCT2 = 2
LINCT3 = 3
C
NUM3 = NUM3 + 1
C
C CARRY?
IF(NUM3 LE 15) GO TO 170
NUM3 = 0
NUM2 = NUM2 + 1
C
C CARRY?
IF(NUM2 LE 15) GO TO 160
NUM2 = 0
NUM1 = NUM1 + 1
IF(NUM1 GE 16) STOP '**** ERROR, PROGRAM EXCEEDS FFFF'
IF((NUM1 GE 8) AND (NUM2 GE 8))
1 PAUSE '**** WARNING, PROGRAM EXCEEDS 87FF, (CR) TO CONTINUE'
PGMCTR(1) = HEXTBL(NUM1 + 1)
PGMCTR(2) = HEXTBL(NUM2 + 1)
160 PGMCTR(3) = HEXTBL(NUM3 + 1)
170
180 CONTINUE
GO TO 600
C
C BLANK OUT END OF LAST LINE
200 NOMORE = LINCT3 + 1
DO 250 JJ = NOMORE,48
250 LIN(JJ) = BLANK
WRITE(12,101) PGMCTR,LIN
GO TO 600
C
C PC MODIFICATION, ZERO FILL TO PC (FROM "BSS" AND "/")
C COMPUTE NEW LINE NUMBER (HEX(1)-HEX(3)) AND COUNTERS
500 PCTEMP = PC
CALL DECHEX(PCTEMP,HEX)
NUM1 = HEX(1) - ASCII0
NUM2 = HEX(2) - ASCII0
NUM3 = HEX(3) - ASCII0

```

```

      IF(NUM1 GT 9) NUM1 = HEX(1) - ASCIIA + 10
      IF(NUM2 GT 9) NUM2 = HEX(2) - ASCIIA + 10
      IF(NUM3 GT 9) NUM3 = HEX(3) - ASCIIA + 10
C
C CALCULATE DIFFERENCE TO NEW LOCATION
      DIFRNC = PC - OLDPC
C
C SIX POSSIBILITIES
C      A) NO MODIFICATION
C      B) READY TO START NEW LINE
C      C) ZERO FILL TO THE MIDDLE OF THE OUTPUTTED LINE
C      D) ZERO FILL TO THE END OF THE OUTPUTTED LINE
C      E) ZERO FILL TO THE END OF THE OUTPUTTED LINE AND PART OF ANOTHER
C         LINE
C      F) ERROR. PC REQUEST IS LESS THAN CURRENT PC
C
      IF(DIFRNC LT 0) GO TO 700
      IF(DIFRNC EQ 0) GO TO 600
      IF(LINCT1 EQ 1) GO TO 570
C
C ZERO FILL OLD LINE
530      DO 550 I1=1,DIFRNC
          LIN(LINCT1) = BLANK
          LIN(LINCT2) = ASCII0
          LIN(LINCT3) = ASCII0
          IF(LINCT3 GE 48) GO TO 560
          LINCT1 = LINCT1 + 3
          LINCT2 = LINCT2 + 3
          LINCT3 = LINCT3 + 3
550      CONTINUE
          GO TO 600
C
D      TYPE 999, NUM1, NUM2, NUM3, OLDPC, OLDBYT, ENDLIN, DIFRNC, PC
D999      FORMAT(1X, 8(16, 2X))
C
C OUTPUT OLD LINE AND UPDATE COUNTER
560      WRITE(12, 101) PGMCTR, LIN
570      DO 580 I1 = 1, 3
580      PGMCTR(I1) = HEX(I1)
C
C ZERO FILL THE BEGINNING OF THE NEW LINE
      LINCT1 = 1
      LINCT2 = 2
      LINCT3 = 3
      DIFRNC = HEX(4) - ASCII0
      IF(DIFRNC GT 9) DIFRNC = HEX(4) - ASCIIA + 10
      IF(DIFRNC EQ 0) GO TO 600
      DO 590 I1=1,DIFRNC
          IF(LINCT3 GT 48) GO TO 600
          LIN(LINCT1) = BLANK
          LIN(LINCT2) = ASCII0
          LIN(LINCT3) = ASCII0
          LINCT1 = LINCT1 + 3
          LINCT2 = LINCT2 + 3
          LINCT3 = LINCT3 + 3
590
600      OLDPC = PC
          OLDBYT = BYTNUM
          RETURN
700      TYPE 701, PC, OLDPC
701      FORMAT(1X, '*** ERROR ***. PC REQUEST OF ', I5, ' IS LESS THAN THE
          1 CURRENT PROGRAM COUNTER ', I5)
          STOP
          END
C
C
C

```

```

SUBROUTINE SRCLST(THISPC)
IMPLICIT INTEGER (A-Z)

C
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
LOGICAL*1 DEFFLG, REGFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
LOGICAL*1 PGMCTR, DUMMY2, HXCODE(4), LIN

C
COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4,10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, BYTFLG
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4 ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5 V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2 VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OPFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
COMMON /LOCAL/ NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),
1 DUMMY2, OLDDPC, OLDBYT, LIN(48)

C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)

C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))

C
C IF THE "LIST" DIRECTIVE IS SET, THIS ROUTINE WILL OUTPUT THE INPUT LINE,
C LINE STATUS AND TARGET CODE.
C
C OBTAIN HEX VALUE OF PC
CALL DECHEX(THISPC, HEX)

C
C IS THIS LINE A COMMENT?
IF(.NOT. COMNTS) GO TO 5
WRITE(13,200) LINENO, LINE
200 FORMAT(1X, I5, 40X, 72A1)
RETURN

C
DO 10 I1 = 1, 4
10 HXCODE(I1) = HEXCCD(I1,1)
N1 = 61
15 N1 = N1 - 1
IF(N1 .LE. 1) GO TO 20
IF(OPRBUF(N1) .EQ. BLANK) GO TO 15
20 WRITE(13,100) LINENO, THISPC, HEX, LNSTAT, HXCODE, LABEL, OPCODE,
1 (OPRBUF(K1), K1=1, N1)
100 FORMAT(1X, I5, 3X, I5, 2X, 4A1, 6X, I3, 8X, 4A1, 5X, 6A1, 1X, 4A1, 4X, 60A1)
IF(WRDNUM .EQ. 1) RETURN
DO 30 I1 = 2, WRDNUM
WRITE(13,25) (HEXCOD(J1, I1), J1 = 1, 4)
30 CONTINUE
25 FORMAT(37X, 4A1)
RETURN
END

```



```

C
C
C
SUBROUTINE ERRORT(THISPC)
IMPLICIT INTEGER (A-Z)
C
LOGICAL*1 LABEL, OPCODE, OPRBUF, HEX, HEXCOD, HEXTBL, LINE
LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
LOGICAL*1 LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
LOGICAL*1 DEFFLG, REGFLG, BYTFLG, LABFLG, INSFLG, OPFLG, RESFLG,
1 DIRFLG, ENDFLG, COMNTS, EXTRA, LSTFLG
LOGICAL*1 PGMCTR, DUMMY2, HXCODE(4), LIN
C
COMMON /ASC/ LINE(72), HEX(4), HEXCOD(4, 10)
COMMON /ERRFLG/ LABERR, DIRERR, OPERR, TYPERR, INSTER, DUMMY
COMMON /NAMFLG/ DEFFLG, LABFLG, REGFLG, BYTFLG
COMMON /LINFLG/ INSFLG, OPFLG, RESFLG, DIRFLG, ENDFLG, COMNTS, EXTRA,
1 LSTFLG
COMMON /ASSEMB/ PASS, REGSRC, REGDES, REGCOD(2), PC, LINENO, ADDRES,
1 MAJVAL, COMENT, IDSTAT, LNSTAT, ERRNUM, FMT, DIRNUM,
2 OPER1, OPER2, ER, ERLABL, WRDNUM, BYTNUM, ERROR
COMMON /DATUM/ FIRST, SECOND, HEXTBL(16)
COMMON /GIRLCH/ EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST,
1 LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH,
2 ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
3 NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST,
4 ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U,
5 V, W, X, Y, Z
COMMON /GIRL/ STRING, TOKEN, ITEM, BUFPOS, LENID, LABELS, OPCODE, STOP,
1 ITMTYP, VALUE, SYMBOL, REG, REGSTR, REGATS, REGPLS,
2 VARABL, NUMBER, LITRAL, LOGOR, LOGAND, LOGNOT, MODULO,
3 OFFIN, DIRECT, START, TOKODE
COMMON /ASCII/ BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
COMMON /LOCAL/ NUM1, NUM2, NUM3, LINCT1, LINCT2, LINCT3, PGMCTR(5),
1 DUMMY2, OLDP, OLD, BYT, LIN(48)
C
DIMENSION LABEL(6), OPCODE(4), OPRBUF(60)
C
EQUIVALENCE (LINE(1), LABEL(1))
EQUIVALENCE (LINE(8), OPCODE(1))
EQUIVALENCE (LINE(13), OPRBUF(1))
C
DATA TEST /0/
C
LNSTAT = 0
IF(LABERR) LNSTAT = 2
IF(INSTER) LNSTAT = LNSTAT + 4
IF(OPERR) LNSTAT = LNSTAT + 8
IF(LNSTAT EQ 0) RETURN
C
C OBTAIN HEX VALUE OF PC
CALL DECHEX(THISPC, HEX)
C
DO 10 I1 = 1, 4
HXCODE(I1) = HEXCOD(I1, 1)
IF(TEST EQ 1) GO TO 70
TYPE 50
FORMAT(/, 1X, ' **** ERROR ****')
TYPE 60
FORMAT(' LINE NO ADDRESS LINE STATUS OBJ CODE LABEL
1 INSTR OP FIELD')
TEST = 1
70 TYPE 100, LINENO, THISPC, HEX, LNSTAT, HXCODE, LABEL, OPCODE, OPRBUF
100 FORMAT(1X, 15, 3X, 15, 2X, 4A1, 6X, 13, 7X, 4A1, 6X, 6A1, 1X, 4A1, 4X, 60A1)
RETURN
END

```

```

C
C
C
      SUBROUTINE TWOCMP(HEX)
      IMPLICIT INTEGER (A-Z)
C
      LOGICAL*1 HEX(4), HEXTBL, NEG
      LOGICAL*1 BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z, ASCHAR
C
      COMMON /DATUM/  FIRST, SECOND, HEXTBL(16)
      COMMON /ASCII/  BLANK, ASCII0, ASCII9, ASCII1A, ASCII1F, ASCII1Z
C
      DIMENSION TEMP(4)
C
C THIS ROUTINE CONVERTS A HEX NUMBER OF FOUR ASCII DIGITS TO ITS
C 2'S COMPLEMENT FORM IN ASCII CODE
C PLACE INTEGER EQUIVALENT INTO "TEMP"
      DO 30 N1 = 1, 4
      VAL = HEX(N1) - ASCII0
      IF(VAL .GT. 9) VAL = HEX(N1) - ASCII1A + 10
      TEMP(N1) = 15 - VAL
30    CONTINUE
      TEMP(4) = TEMP(4) + 1
C
C CHECK FOR OVERFLOW
      N1 = 5
40    N1 = N1 - 1
      IF(TEMP(N1) .LE. 15) GO TO 50
      TEMP(N1) = 0
      IF(N1 .LE. 1) GO TO 50
      TEMP(N1-1) = TEMP(N1-1) + 1
      GO TO 40
C
50    DO 60 N1 = 1, 4
      HEX(N1) = HEXTBL(TEMP(N1) + 1)
60    CONTINUE
      RETURN
      END

```

APPENDIX C
VARIABLES IN LABELED COMMON

/ASSEMB/

- PASS - Assembler pass number. It is set to either 1 or 2.
- REGSRC - Source field workspace register number. Value range 0-15.
- REGDES - Destination field workspace register number. Value range 0-15.
- REGCOD() - Source (1) and destination (2) register codes. Allowable values:
 Ri = 0
 *Ri = 1
 @Ri = 2
 *Ri+ = 3
- PC - Current value of Program Counter.
 It is reset at the beginning of pass two and has a default value of 100_{16} (256_{10}). It is user modifiable (see directives AORG, /, BSS).
- LINENO - Input line number.
- ADDRES - Used by Subroutine ADDNAM to return an address value for a requested identifier. Value is zero if identifier is not yet defined. It is used by Subroutine OPFLD to return the value of a complete operand field.
- MAJVAL - Output from Subroutine OPFLD. It is used by Subroutine ITMSCN to check that an operand field is semantically correct. Refer to Tables 1 and 2.
- COMENT - Location of the semicolon in the input line. It indicates where comments begin for that line. Value range is 1 to 60.
- IDSTAT - Output from Subroutine ADDNAM. It holds the status of the retrieved identifier. Allowable values include various combinations of the following:
 1 - Identifier has been defined
 2 - Identifier is a register
 4 - Identifier is multipli-defined
 8 - Identifier is a halfword (byte).

PRECEDING PAGE BLANK-NOT FL

LNSTAT - Status of the input statement. Allowable values include any cumulative combination of the following:

- 2 - Label error
- 4 - Instruction or directive error
- 8 - Operand error

ERRNUM - Reserved for future program enhancement.

FMT - Output from Subroutine INSTRU. At the successful completion of a search of the instruction-directive tree, FMT contains the operand format number for the requested instruction. Refer to Tables 1 and 2.

DIRNUM - Output from Subroutine INSTRU. At the successful completion of a search of the instruction-directive tree, DIRNUM contains the directive number which uniquely defines the directive. Refer to Table 3.

OPER1 and OPER2 - Output from Subroutine INSTRU. At the successful completion of a search of the instruction-directive tree, OPER1 and OPER2 contain the ASCII equivalent of the instruction skeleton.

ER - State pointer for the identifier tree.

ERLABL - State pointer for the identifier tree. It points to the last source node in the string describing the current label. Its link is always "STOP." Refer to discussion on the identifier tree.

WRDNUM - Number of words required by either an instruction or a directive. It is computed during the first (pass) call to Subroutine ITMSCN. Value range is:

Instructions	Directives
1-3	0-10

BYTNUM - Number of bytes required by either an instruction or a directive. It is computed during the first (pass) call to Subroutine ITMSCN and is an input to Subroutine ABSOUT. Value range is:

Instructions	Directives
2-6	0-20

ERROR - Reserved for future program enhancement

/GIRLCH/ - EXCLAM, QUOTE, POUND, DOLLAR, PERCNT, AMPERS, APOST, LPAR, RPAR, STAR, PLUS, COMMA, MINUS, PERIOD, SLASH, ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, COLON, SCOLON, LSTHAN, EQUALS, GTTHAN, QUEST, ATSIGN, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

These variables are defined within GIRL² to represent the particular character as named.

/ERRFLG/ - The following logical*1 variables have the indicated meanings when set to .TRUE.:

LABERR - Label error such as multiple definition.

DIRERR - Not used in present version.

OPERR - Error in operand field.

TYPERR - Reserved for future program enhancement.

INSTER - Instruction name or directive name error.

DUMMY - Filler byte to make this common block contain an even number of bytes.

/NAMFLG/ - The following flags indicate label status. If the status of an identifier includes any of the flags, then that identifier has the following properties:

DEFFLG - Identifier has been defined and given an address by virtue of placement in the label field.

LABFLG - Not used in present version.

REGFLG - Identifier is a register.

MLTFLG - Identifier appears in the label field more than once.

BYTFLG - Identifier is halfword size.

DUMB - Filler byte to make this common block contain an even number of blocks.

/LINFLG/

INSFLG - Not used in present version.

OPFLG - Not used in present version.

RESFLG - Reserved for future program enhancement.

DIRFLG - Not used in present version.

ENDFLG - Set to .TRUE. when END directive is encountered.

COMNTS - Set to .TRUE. if entire input line is a comment statement.

- EXTRA - Input to Subroutine ITMSCN from Subroutine OPFLD. It is set to .TRUE. If there is a memory reference in an operand field, thus requiring an extra machine code word on output.
- LSTFLG - Set to .TRUE. if LIST directive is encountered.

APPENDIX D
SAMPLE SET OF INSTRUCTIONS AND DIRECTIVES

```

LIST ;
A R1,R2;
JIM AB R2,R1;
ABS R1;
AI R1,>ABC;
ANDI R2,>ABC;
UNL ;
ABS R1;
AI R1,>ABC;
ANDI R2,>ABC;
b R1;
BILL BL R2;
BLWP R2;
JOE BSS >A;
SID BYTE >10,120;
C R2,R1;
BYTE 'A';
BYTE 'B';
BYTE 'C';
BYTE 'D';
CB R1,R2;
CI R1,>7FF;
CLR R2;
COC R2,R1;
CZC R2,R1;
WALTER DATA 'AB',>23;
DEC R1;
DECT R2;
DIV R2,R1;
DXOP ;
JOAN EQU 3+5*6;
CNTHIA EQU JOAN+10+6;
AI R1,JOAN;
AI R1,RAE;
SHIRA EVEN ;
CAROL BYTE 'A',*B*,>EF;
RAE EVEN ;
IDLE ;
IN R1;
INC R2;
INCT R2;
INV R1;
JEQ 300;
JGT @300;
JH 300;
JHE 300;
JL 300;
JLE 300;
JLT 300;
JMP 300;
JNC 300;
JNE 300;
JNO 300;
JOC 300;
JOP 300;
GRTF LDCR R1,B;
PLKJ LI R1,>ABC;
FHODJ LIMI >ABC;

```



```

PYTRV  LWPI  >ABC;
HJFG   MOV   R1,R2;
SFD     MOVB  R1,R2;
WR      MPY   R2,R1;
XZC     NEG   R1;
WRGY    NOP   ;
LKUY    ORI   R1,>ABC;
PGTYR   OUT   R1;
RT      ;
LDFWG   RTWP  ;
WTUEY   S     R1,R2;
GAZXC   SB    R2,R1;
PLMN    SBO   14;
EFVBG   SBZ   20;
DCFE    SETO  R1;
POIJ    SLA   R1,12;
QWER    SOC   R1,R2;
WALT    SOCB  R2,R1;
KLEE    SRA   R2,10;
TJUF    SRC   R1,12;
PETW    SRL   R2,8;
SDWER    STCR R2,15;
WDFVE   STST  R1;
EVEFR   STWP  R1;
        SWPB  R1;
PLKJH   SZC   R2,R1;
SKJHY   SZCB  R2,R1;
LADWT   TB    18;
        TEXT 'EXAMPLE';
        EVEN ;
        END  ;
WERE    X     R2;
QSPK    XOP   R2;
NVBCW   XOR   R1,R2;
        /     >60A;
        IN    R1;
        INC   R2;
        INCT  R2;
        INV   R1;
        JEQ   300;
        JGT   @300;
        JH    300;
        JHE   300;
        JL    300;
        JLE   300;
        JLT   300;
        END  ;

```

APPENDIX E
SOURCE AND IDENTIFIER LISTING GENERATED FROM
APPENDIX A AS INPUT

LINE NO.	ADDRESS	LINE STATUS	OBJ CODE	LABEL	INSTR.	OP FIELD
1					LIST ;	
2	256 0100	0	A0B1		A	R1, R2;
3	258 0102	0	B042	JIM	AB	R2, R1;
4	260 0104	0	0741		ABS	R1;
5	262 0106	0	0221		AI	R1, >ABC;
			0ABC			
6	266 010A	0	0242		ANDI	R2, >ABC;
			0ABC			
7					UNL ;	
8	270 010E	0	0741		ABS	R1;
9	272 0110	0	0221		AI	R1, >ABC;
			0ABC			
10	276 0114	0	0242		ANDI	R2, >ABC;
			0ABC			
11	280 011B	0	0441		B	R1;
12	282 011A	0	06B2	BILL	BL	R2;
13	284 011C	0	0402		BLWP	R2;
14	286 011E	0		JOE	BSS	>A;
15	296 012B	0	10B0	SID	BYTE	>10, 12B;
16	298 012A	0	B042		C	R2, R1;
17	300 012C	0	41		BYTE	'A';
18	301 012D	0	42		BYTE	'B';
19	302 012E	0	43		BYTE	'C';
20	303 012F	0	44		BYTE	'D';
21	304 0130	0	90B1		CB	R1, R2;
22	306 0132	0	02B1		CI	R1, >7FF;
			07FF			
23	310 0136	0	04C2		CLR	R2;
24	312 013B	0	2042		CDC	R2, R1;
25	314 013A	0	2442		CZC	R2, R1;
26	316 013C	0	4142	WALTER	DATA	'AB', >23;
			0023			
27	320 0140	0	0601		DEC	R1;
28	322 0142	0	0642		DECT	R2;
29	324 0144	0	3C42		DIV	R2, R1;
30					DXOP ;	
31				JOAN	EQU	3+5*6;
32				CNTHIA	EQU	JOAN+10+6;
33	326 0146	0	0221		AI	R1, JOAN;
			0030			
34	330 014A	0	0221		AI	R1, RAE;
			0151			
35				SHIRA	EVEN ;	
36	334 014E	0	4142	CAROL	BYTE	'A', *B*, >EF;
			EF			
37	337 0151	0	00	RAE	EVEN	;
38	338 0152	0	0340		IDLE	;
39	340 0154	0	2C41		IN	R1;
40	342 0156	0	05B2		INC	R2;
41	344 015B	0	05C2		INCT	R2;
42	346 015A	0	0541		INV	R1;
43	348 015C	0	13E7		JEQ	300;
44	350 015E	0	15E6		JGT	@300;
45	352 0160	0	1BE5		JH	300;
46	354 0162	0	14E4		JHE	300;
47	356 0164	0	1AE3		JL	300;
48	358 0166	0	12E2		JLE	300;

49	360	016B	0	11E1	JLT	300;
50	362	016A	0	10E0	JMP	300;
51	364	016C	0	17DF	JNC	300;
52	366	016E	0	16DE	JNE	300;
53	368	0170	0	19DD	JNO	300;
54	370	0172	0	18DC	JOC	300;
55	372	0174	0	1CDB	JOP	300;
56	374	0176	0	3201	GRTF, LDCR	R1, B;
57	376	0178	0	0201	PLKJ, LI	R1, >ABC;
				0ABC		
58	380	017C	0	0300	FHGDJ, LIM1	>ABC;
				0ABC		
59					LIST ;	
60	384	0180	0	02E0	PYTRV, LWPI	>ABC;
				0ABC		
61	388	0184	0	C0B1	HJFG, MOV	R1, R2;
62	390	0186	0	D0B1	SFD, MOVB	R1, R2;
63	392	0188	0	3842	WR, MPY	R2, R1;
64	394	018A	0	0501	XZC, NEG	R1;
65	396	018C	0	1000	WRQY, NOP	;
66	398	018E	0	0261	LKUY, ORI	R1, >ABC;
				0ABC		
67	402	0192	0	2CB1	PQTYR, OUT	R1;
68	404	0194	0	045B	RT	;
69	406	0196	0	03B0	LDFWG, RTWP	;
70	408	0198	0	60B1	WTUEY, S	R1, R2;
71	410	019A	0	7042	GAZXC, SB	R2, R1;
72	412	019C	0	1D0E	PLMN, SBO	14;
73	414	019E	0	1E14	EFVBC, SBZ	20;
74	416	01A0	0	0701	DCFE, SETO	R1;
75	418	01A2	0	0AC1	POIJ, SLA	R1, 12;
76	420	01A4	0	E0B1	QWER, SOC	R1, R2;
77	422	01A6	0	F042	WALT, SOCB	R2, R1;
78	424	01A8	0	0BA2	KLEE, SRA	R2, 10;
79	426	01AA	0	0BC1	TJUF, SRC	R1, 12;
80	428	01AC	0	09B2	PETW, SRL	R2, B;
81	430	01AE	0	37C2	SDWER, STCR	R2, 15;
82	432	01B0	0	02C1	WDFVE, STST	R1;
83	434	01B2	0	02A1	EVEFR, STWP	R1;
84	436	01B4	0	06C1	SWPB	R1;
85	438	01B6	0	4042	PLKJH, SZC	R2, R1;
86	440	01B8	0	5042	SKJHY, SZCB	R2, R1;
87	442	01BA	0	1F12	LADWT, TB	1B;
88	444	01BC	0	455B	TEXT	'EXAMPLE';
				414D		
				504C		
				45		
89	451	01C3	0	00	EVEN	;
90	452	01C4	0		END	;

**** IDENTIFIER LISTING ****

NAME	STATUS ADDRESS	REG	UNDEF	MULT DEF	BYTE
R0	0 0000	X			
R1	1 0001	X			
R2	2 0002	X			
R3	3 0003	X			
R4	4 0004	X			
R5	5 0005	X			
R6	6 0006	X			
R7	7 0007	X			
R8	8 0008	X			
R9	9 0009	X			
R10	10 000A	X			
R11	11 000B	X			
R12	12 000C	X			
R13	13 000D	X			
R14	14 000E	X			
R15	15 000F	X			
JIM	258 0102				
BILL	282 011A				
JOE	286 011E				
SID	296 0128				X
JOAN	48 0030				
CNTHIA	64 0040				
RAE	337 0151				
SHIRA	334 014E				
CAROL	334 014E				X
RAE	337 0151				
GRTF	374 0176				
PLKJ	376 0178				
FHGDJ	380 017C				
PYTRV	384 0180				
HJFG	388 0184				
SFD	390 0186				
WR	392 0188				
XZC	394 018A				
WRQY	396 018C				
LKUY	398 018E				
PGTYR	402 0192				
LDFWG	406 0196				
WTUEY	408 0198				
GAZXC	410 019A				
PLMN	412 019C				
EFVBC	414 019E				
DCFE	416 01A0				
POIJ	418 01A2				
QWER	420 01A4				
WALT	422 01A6				
KLEE	424 01A8				
TJUF	426 01AA				
PETW	428 01AC				
SDWER	430 01AE				
WDFVE	432 01B0				
EVEFR	434 01B2				
PLKJH	438 01B6				
SKJHY	440 01B8				
LADWT	442 01BA				

APPENDIX F
MACHINE CODE GENERATED FROM APPENDIX A AS INPUT

0100: A0 B1 B0 42 07 41 02 21 0A BC 02 42 0A BC 07 41
 0110: 02 21 0A BC 02 42 0A BC 04 41 06 B2 04 02 00 41
 0120: 00 00 00 00 00 00 00 00 10 80 80 42 41 42 43 44
 0130: 90 B1 02 B1 07 FF 04 C2 20 42 24 42 41 42 00 23
 0140: 06 01 06 42 3C 42 02 21 00 30 02 21 01 51 41 42
 0150: EF 00 03 40 2C 41 05 B2 05 C2 05 41 13 E7 15 E6
 0160: 1B E5 14 E4 1A E3 12 E2 11 E1 10 E0 17 DF 16 DE
 0170: 19 DD 1B DC 1C DB 32 01 02 01 0A BC 03 00 0A BC
 0180: 02 E0 0A BC C0 B1 D0 B1 38 42 05 01 10 00 02 61
 0190: 0A BC 2C B1 04 5B 03 80 60 B1 70 42 1D 0E 1E 14
 01A0: 07 01 0A C1 E0 B1 F0 42 08 A2 0B C1 09 B2 37 C2
 01B0: 02 C1 02 A1 06 C1 40 42 50 42 1F 12 45 5B 41 4D
 01C0: 50 4C 45 00

REFERENCES

1. "9900 Family Systems Design and Data Book," Texas Instruments, Inc., First Edition, Houston, Texas (1978).
2. Berkowitz, S., "Graph Information Retrieval Language; Programing Manual for FORTRAN Complement; Revision One," DTNSRDC Report 76-0085 (Feb 1976).
3. Zaritsky, I., "GIRS (Graph Information Retrieval System) User Manual," DTNSRDC Report 79/036 (Apr 1979).

INITIAL DISTRIBUTION

Copies		Copies	Code	Name
1	CHONR/430D	1	1826	L. Culpepper
1	NRL	1	1828	C. Godfrey
		1	1828	W. Gorham, Jr.
1	NSWC	1	184	J. Schot
1	NUSC	1	184.1	H. Feingold
1	NOSC	1	1843	H. Haussling
		1	1844	S. Dhir
1	NAVSUP/0431C, G. Bernstein	1	1844	J. McKee
		1	185	T. Corin
2	NAVSEA	1	1850	A. Cinque
	1 SEA 312	1	1851	J. Brainin
	1 SEA 612	1	1854	H. Sheridan
1	Rome Air Development Center	1	1855	R. Brengs
12	DTIC	1	187	R. Ploe
		1	187	M. Zubkoff
2	Texas Instruments, Inc.	1	189	G. Gray
	1 James B. Allen			
	CENTER DISTRIBUTION	10	5211.1	Reports Distribution
		1	522.1	Unclassified Lib (C)
		1	522.2	Unclassified Lib (A)

Copies	Code	Name
1	1576	C. Bell
1	18	G. Gleissner
1	1802.2	F. Frenkiel
1	1803	S. Rainey
1	1804	L. Avrunin
1	1805	E. Cuthill
1	1806	R. Santamaria
2	1809.3	D. Harris
1	182	A. Camara
1	1821	D. Jefferson
1	1822	T. Rhodes
1	1824	S. Berkowitz
1	1824	J. Carlberg
1	1824	J. Garner
1	1824	P. Marques
20	1824	I. Zaritsky

PRECEDING PAGE BLANK-NOT FILMED

DAT
ILMI